MLPACK

1.0.8

Generated by Doxygen 1.8.6

Sat Apr 12 2014 22:41:30

Contents

Chapter 1

MLPACK Documentation

1.1 Introduction

MLPACK is an intuitive, fast, scalable C++ machine learning library, meant to be a machine learning analog to LAPACK. It aims to implement a wide array of machine learning methods and function as a "swiss army knife" for machine learning researchers. The MLPACK development website can be found at `http://mlpack.org`.

MLPACK uses the Armadillo C++ matrix library (`http://arma.sourceforge.net`) for general matrix, vector, and linear algebra support. MLPACK also uses the program_options, math_c99, and unit_test_framework components of the Boost library; in addition, LibXml2 is used.

1.2 How To Use This Documentation

This documentation is API documentation similar to Javadoc. It isn't necessarily a tutorial, but it does provide detailed documentation on every namespace, method, and class.

Each MLPACK namespace generally refers to one machine learning method, so browsing the list of namespaces provides some insight as to the breadth of the methods contained in the library.

To generate this documentation in your own local copy of MLPACK, you can simply use Doxygen, from the root directory (`/mlpack/trunk/`):

```
$ doxygen
```

1.3 Executables

MLPACK provides several executables so that MLPACK methods can be used without any need for knowledge of C++. These executables are all self-documented, and that documentation can be accessed by running the executables with the '-h' or '–help' flag.

A full list of executables is given below:

allkfn, allknn, emst, gmm, hmm_train, hmm_loglik, hmm_viterbi, hmm_generate, kernel_pca, kmeans, lars, linear_-regression, local_coordinate_coding, mvu, nbc, nca, pca, radical, sparse_coding

1.4 Tutorials

A few short tutorials on how to use MLPACK are given below.

- **Building MLPACK From Source** (p. 5)
- **Matrices in MLPACK** (p. 13)
- **MLPACK Input and Output** (p. 9)
- **MLPACK Timers** (p. 17)
- **Simple Sample MLPACK Programs** (p. 15)
- **mlpack version information** (p. 19)

Tutorials on specific methods are also available.

- **NeighborSearch tutorial (k-nearest-neighbors)** (p. 55)
- **Linear/ridge regression tutorial (linear_regression)** (p. 49)
- **RangeSearch tutorial (range_search)** (p. 61)
- **Density Estimation Tree (DET) tutorial** (p. 21)
- **EMST Tutorial** (p. 27)
- **K-Means tutorial (kmeans)** (p. 39)
- **Fast max-kernel search tutorial (fastmks)** (p. 31)

1.5 Methods in MLPACK

The following methods are included in MLPACK:

- Density Estimation Trees - **mlpack::det::DTree** (p. 145)
- Euclidean Minimum Spanning Trees - **mlpack::emst::DualTreeBoruvka** (p. 172)
- Gaussian Mixture Models (GMMs) - **mlpack::gmm::GMM** (p. 208)
- Hidden Markov Models (HMMs) - **mlpack::hmm::HMM** (p. 221)
- Kernel PCA - **mlpack::kpca::KernelPCA** (p. 271)
- K-Means Clustering - **mlpack::kmeans::KMeans** (p. 260)
- Least-Angle Regression (LARS/LASSO) - **mlpack::regression::LARS** (p. 429)
- Local Coordinate Coding - **mlpack::lcc::LocalCoordinateCoding** (p. 274)
- Locality-Sensitive Hashing - **mlpack::neighbor::LSHSearch** (p. 307)
- Naive Bayes Classifier - **mlpack::naive_bayes::NaiveBayesClassifier** (p. 292)
- Neighborhood Components Analysis (NCA) - **mlpack::nca::NCA** (p. 296)
- Principal Components Analysis (PCA) - **mlpack::pca::PCA** (p. 407)

- RADICAL (ICA) - **mlpack::radical::Radical** (p. 411)
- Simple Least-Squares Linear Regression - **mlpack::regression::LinearRegression** (p. 435)
- Sparse Coding - **mlpack::sparse_coding::SparseCoding** (p. 450)
- Tree-based neighbor search (AllkNN, AllkFN) - **mlpack::neighbor::NeighborSearch** (p. 317)
- Tree-based range search - **mlpack::range::RangeSearch** (p. 415)

1.6 Final Remarks

This software was written in the FASTLab (`http://www.fast-lab.org`), which is in the School of Computational Science and Engineering at the Georgia Institute of Technology.

MLPACK contributors include:

- Ryan Curtin `gth671b@mail.gatech.edu`
- James Cline `james.cline@gatech.edu`
- Neil Slagle `nslagle3@gatech.edu`
- Matthew Amidon `mamidon@gatech.edu`
- Vlad Grantcharov `vlad321@gatech.edu`
- Ajinkya Kale `kaleajinkya@gmail.com`
- Bill March `march@gatech.edu`
- Dongryeol Lee `dongryel@cc.gatech.edu`
- Nishant Mehta `niche@cc.gatech.edu`
- Parikshit Ram `p.ram@gatech.edu`
- Rajendran Mohan `rmohan88@gatech.edu`
- Trironk Kiatkungwanglai `trironk@gmail.com`
- Patrick Mason `patrick.s.mason@gmail.com`
- Chip Mappus `cmappus@gatech.edu`
- Hua Ouyang `houyang@gatech.edu`
- Long Quoc Tran `tqlong@gmail.com`
- Noah Kauffman `notoriousnoah@gmail.com`
- Guillermo Colon `gcolon7@mail.gatech.edu`
- Wei Guan `wguan@cc.gatech.edu`
- Ryan Riegel `rriegel@cc.gatech.edu`
- Nikolaos Vasiloglou `nvasil@ieee.org`
- Garry Boyer `garryb@gmail.com`
- Andreas Löf `andreas.lof@cs.waikato.ac.nz`

- Marcus Edel marcus.edel@fu-berlin.de
- Mudit Raj Gupta mudit.raaj.gupta@gmail.com
- Sumedh Ghaisas sumedhghaisas@gmail.com

Chapter 2

Building MLPACK From Source

2.1 Introduction

MLPACK uses CMake as a build system and allows several flexible build configuration options. One can consult any of numerous CMake tutorials for further documentation, but this tutorial should be enough to get MLPACK built and installed.

2.2 latest mlpack build Download latest mlpack build from here : mlpack-1.0.8

2.3 Creating Build Directory

Once the MLPACK source is unpacked, you should create a build directory.

```
$ cd mlpack-1.0.8
$ mkdir build
```

The directory can have any name, not just 'build', but 'build' is sufficient enough.

2.4 Dependencies of MLPACK

MLPACK depends on the following libraries, which need to be installed on the system and have headers present:

- Armadillo >= 3.6.0 (with LAPACK support)
- LibXML2 >= 2.6.0
- Boost (math_c99, program_options, unit_test_framework, random)

In Ubuntu and Debian, you can get all of these dependencies through apt:

```
# apt-get install libboost-math-dev libboost-program-options-dev
  libboost-random-dev libboost-test-dev libxml2-dev libarmadillo-dev
```

If you are using an Ubuntu version older than 13.10 ("Saucy Salamander") or Debian older than Jessie, you will have to compile Armadillo from source. See the README.txt distributed with Armadillo for more information.

On Fedora, Red Hat, or CentOS, these same dependencies can be obtained via yum:

```
# yum install boost-devel boost-random boost-test boost-program-options
  boost-math libxml2-devel armadillo-devel
```

On Red Hat Enterprise Linux 5 and older (as well as CentOS 5), the Armadillo version available is too old and must be compiled by hand. The same applies for Fedora 16 and older.

2.5 Configuring CMake

Running CMake is the equivalent to running `./configure` with autotools. If you run CMake with no options, it will configure the project to build with debugging symbols and profiling information:

```
$ cd build
$ cmake ../
```

You can specify options to compile without debugging information and profiling information (i.e. as fast as possible):

```
$ cd build
$ cmake -D DEBUG=OFF -D PROFILE=OFF ../
```

The full list of options MLPACK allows:

- DEBUG=(ON/OFF): compile with debugging symbols (default ON)
- PROFILE=(ON/OFF): compile with profiling symbols (default ON)
- ARMA_EXTRA_DEBUG=(ON/OFF): compile with extra Armadillo debugging symbols (default OFF)

Each option can be specified to CMake with the '-D' flag. Other tools can also be used to configure CMake, but those are not documented here.

2.6 Building MLPACK From Source

Once CMake is configured, building the library is as simple as typing 'make'. This will build all library components as well as 'mlpack_test'.

```
$ make
Scanning dependencies of target mlpack
[  1%] Building CXX object
src/mlpack/CMakeFiles/mlpack.dir/core/optimizers/aug_lagrangian/aug_lagrangian_test_functions.cpp.o
<...>
```

You can specify individual components which you want to build, if you do not want to build everything in the library:

```
$ make pca allknn allkfn
```

If the build fails and you cannot figure out why, register an account on Trac and submit a ticket and the MLPACK developers will quickly help you figure it out:

`http://mlpack.org/`

Alternately, MLPACK help can be found in IRC at #mlpack on irc.freenode.net.

2.7 Installing MLPACK

If you wish to install MLPACK to /usr/include/mlpack/ and /usr/lib/ and /usr/bin/, once it has built, make sure you have root privileges (or write permissions to those two directories), and simply type

```
# make install
```

You can now run the executables by name; you can link against MLPACK with -lmlpack, and the MLPACK headers are found in /usr/include/mlpack/.

Chapter 3

MLPACK Input and Output

3.1 Introduction

MLPACK provides the following:

- **mlpack::Log** (p. 279), for debugging / informational / warning / fatal output
- **mlpack::CLI** (p. 132), for parsing command line options

Each of those classes are well-documented, and that documentation should be consulted for further reference.

3.2 Simple Logging Example

MLPACK has four logging levels:

- Log::Debug
- Log::Info
- Log::Warn
- Log::Fatal

Output to Log::Debug does not show (and has no performance penalty) when MLPACK is compiled without debugging symbols. Output to Log::Info is only shown when the program is run with the –verbose (or -v) flag. Log::Warn is always shown, and Log::Fatal will halt the program, when a newline is sent to it.

Here is a simple example, and its output:

```
#include <mlpack/core.hpp>

using namespace mlpack;

int main()
{
  Log::Debug << "Compiled with debugging symbols." << std::endl;

  Log::Info << "Some test informational output." << std::endl;

  Log::Warn << "A warning!" << std::endl;
```

```
  Log::Fatal << "Program has crashed." << std::endl;

  Log::Warn << "Made it!" << std::endl;
}
```

With debugging output and –verbose, the following is shown:

```
$ ./main --verbose
[DEBUG] Compiled with debugging symbols.
[INFO ] Some test informational output.
[WARN ] A warning!
[FATAL] Program has crashed.
```

The last warning is not reached, because Log::Fatal terminates the program.

Without debugging symbols and without –verbose, the following is shown:

```
$ ./main
[WARN ] A warning!
[FATAL] Program has crashed.
```

These four outputs can be very useful for both providing informational output and debugging output for your MLPACK program.

3.3 Simple CLI Example

Through the **mlpack::CLI** (p. 132) object, command-line parameters can be easily added with the PROGRAM_INFO, PARAM_INT, PARAM_DOUBLE, PARAM_STRING, and PARAM_FLAG macros.

Here is a sample use of those macros, extracted from methods/pca/pca_main.cpp.

```
#include <mlpack/core.hpp>

// Document program.
PROGRAM_INFO("Principal Components Analysis", "This program performs principal "
    "components analysis on the given dataset.  It will transform the data "
    "onto its principal components, optionally performing dimensionality "
    "reduction by ignoring the principal components with the smallest "
    "eigenvalues.");

// Parameters for program.
PARAM_STRING_REQ("input_file", "Input dataset to perform PCA on.", "");
PARAM_STRING_REQ("output_file", "Output dataset to perform PCA on.", "");
PARAM_INT("new_dimensionality", "Desired dimensionality of output dataset.",
    "", 0);

using namespace mlpack;

int main(int argc, char** argv)
{
  // Parse commandline.
  CLI::ParseCommandLine(argc, argv);

  ...
}
```

Documentation is automatically generated using those macros, and when the program is run with –help the following is displayed:

```
$ pca --help
Principal Components Analysis

  This program performs principal components analysis on the given dataset.  It
  will transform the data onto its principal components, optionally performing
  dimensionality reduction by ignoring the principal components with the
  smallest eigenvalues.
```

```
Required options:

  --input_file [string]         Input dataset to perform PCA on.
  --output_file [string]        Output dataset to perform PCA on.

Options:

  --help (-h)                   Default help info.
  --info [string]               Get help on a specific module or option.
                                Default value ''.
  --new_dimensionality [int]    Desired dimensionality of output dataset.
                                Default value 0.
  --verbose (-v)                Display informational messages and the full list
                                of parameters and timers at the end of
                                execution.
```

The **mlpack::CLI** (p. 132) documentation can be consulted for further and complete documentation.

Chapter 4

Matrices in MLPACK

4.1 Introduction

MLPACK uses Armadillo matrices for matrix support. Armadillo is a fast C++ matrix library which makes use of advanced template techniques to provide the fastest possible matrix operations.

Documentation on Armadillo can be found on their website:

`http://arma.sourceforge.net/docs.html`

Nonetheless, there are a few further caveats for MLPACK Armadillo usage.

4.2 Column-wise Matrices

Armadillo matrices are stored in a column-major format; this means that on disk, each column is located in contiguous memory.

This means that, for the vast majority of machine learning methods, it is faster to store observations as columns and dimensions as rows. This is counter to most standard machine learning texts!

Major implications of this are for linear algebra. For instance, the covariance of a matrix is typically

$C = X^T X$

but for a column-wise matrix, it is

$C = X X^T$

and this is very important to keep in mind! If your MLPACK code is not working, this may be a factor in why.

4.3 Loading Matrices

MLPACK provides a **data::Load()** (p. 85) and **data::Save()** (p. 87) function, which should be used instead of Armadillo's loading and saving functions.

Most machine learning data is stored in row-major format; a CSV, for example, will generally have one observation per line and each column will correspond to a dimension.

The **data::Load()** (p. 85) and **data::Save()** (p. 87) functions transpose the matrix upon loading, meaning that the following CSV:

```
$ cat data.csv
3,3,3,3,0
3,4,4,3,0
3,4,4,3,0
3,3,4,3,0
3,6,4,3,0
2,4,4,3,0
2,4,4,1,0
3,3,3,2,0
3,4,4,2,0
3,4,4,2,0
3,3,4,2,0
3,6,4,2,0
2,4,4,2,0
```

is actually loaded with 5 rows and 13 columns, not 13 rows and 5 columns like the CSV is written.

This is important to remember!

Chapter 5

Simple Sample MLPACK Programs

5.1 Introduction

On this page, several simple MLPACK examples are contained, in increasing order of complexity.

5.2 Covariance Computation

A simple program to compute the covariance of a data matrix ("data.csv"), assuming that the data is already centered, and save it to file.

```
// Includes all relevant components of MLPACK.
#include <mlpack/core.hpp>

// Convenience.
using namespace mlpack;

int main()
{
  // First, load the data.
  arma::mat data;
  // Use data::Load() which transposes the matrix.
  data::Load("data.csv", data, true);

  // Now compute the covariance.  We assume that the data is already centered.
  // Remember, because the matrix is column-major, the covariance operation is
  // transposed.
  arma::mat cov = data * trans(data) / data.n_cols;

  // Save the output.
  data::Save("cov.csv", cov, true);
}
```

5.3 Nearest Neighbor

This simple program uses the **mlpack::neighbor::NeighborSearch** (p. 317) object to find the nearest neighbor of each point in a dataset using the L1 metric, and then print the index of the neighbor and the distance of it to stdout.

```
#include <mlpack/core.hpp>
#include <mlpack/methods/neighbor_search/neighbor_search.hpp>

using namespace mlpack;
using namespace mlpack::neighbor; // NeighborSearch and NearestNeighborSort
using namespace mlpack::metric; // ManhattanDistance
```

```
int main()
{
  // Load the data from data.csv (hard-coded).  Use CLI for simple command-line
  // parameter handling.
  arma::mat data;
  data::Load("data.csv", data, true);

  // Use templates to specify that we want a NeighborSearch object which uses
  // the Manhattan distance.
  NeighborSearch<NearestNeighborSort, ManhattanDistance> nn(data);

  // Create the object we will store the nearest neighbors in.
  arma::Col<size_t> neighbors;
  arma::vec distances; // We need to store the distance too.

  // Compute the neighbors.
  nn.Search(1, neighbors, distances);

  // Write each neighbor and distance using Log.
  for (size_t i = 0; i < neighbors.n_elem; ++i)
  {
    Log::Info << "Nearest neighbor of point " << i << " is point "
        << neighbors[i] << " and the distance is " << distances[i] << ".\n";
  }
}
```

5.4 Other examples

For more complex examples, it is useful to refer to the main executables:

- methods/neighbor_search/allknn_main.cpp
- methods/neighbor_search/allkfn_main.cpp
- methods/emst/emst_main.cpp
- methods/radical/radical_main.cpp
- methods/nca/nca_main.cpp
- methods/naive_bayes/nbc_main.cpp
- methods/pca/pca_main.cpp
- methods/lars/lars_main.cpp
- methods/linear_regression/linear_regression_main.cpp
- methods/gmm/gmm_main.cpp
- methods/kmeans/kmeans_main.cpp

Chapter 6

MLPACK Timers

6.1 Introduction

MLPACK provides a simple timer interface for the timing of machine learning methods. The results of any timers used during the program are displayed at output by the **mlpack::CLI** (p. 132) object, when –verbose is given:

```
$ allknn -r dataset.csv -n neighbors_out.csv -d distances_out.csv -k 5 -v
<...>
[INFO ] Program timers:
[INFO ]   computing_neighbors: 0.010650s
[INFO ]   loading_data: 0.002567s
[INFO ]   saving_data: 0.001115s
[INFO ]   total_time: 0.149816s
[INFO ]   tree_building: 0.000534s
```

6.2 Timer API

The **mlpack::Timer** (p. 455) class provides three simple methods:

```
void Timer::Start(const char* name);
void Timer::Stop(const char* name);
timeval Timer::Get(const char* name);
```

Each timer is given a name, and is referenced by that name. You can call `Timer::Start()` and `Timer::Stop()` multiple times for a particular timer name, and the result will be the sum of the runs of the timer. Note that `Timer::Stop()` must be called before `Timer::Start()` is called again.

A "total_time" timer is run by default for each MLPACK program.

6.3 Timer Example

Below is a very simple example of timer usage in code.

```
#include <mlpack/core.hpp>

using namespace mlpack;

int main(int argc, char** argv)
{
  CLI::ParseCommandLine(argc, argv);
```

```
  // Start a timer.
  Timer::Start("some_timer");

  // Do some things.
  DoSomeStuff();

  // Stop the timer.
  Timer::Stop("some_timer");
}
```

If the –verbose flag was given to this executable, the resultant time that "some_timer" ran for would be shown.

Chapter 7

mlpack version information

7.1 mlpack versions in code

mlpack provides a couple of convenience macros and functions to get the version of mlpack. More information (and straightforward code) can be found in **src/mlpack/core/util/version.hpp** (p. 641).

The following three macros provide major, minor, and patch versions of mlpack (i.e. for mlpack-x.y.z, 'x' is the major version, 'y' is the minor version, and 'z' is the patch version):

```
__MLPACK_VERSION_MAJOR
__MLPACK_VERSION_MINOR
__MLPACK_VERSION_PATCH
```

In addition, the function **mlpack::util::GetVersion()** (p. 110) returns the mlpack version as a string (for instance, "mlpack 1.0.8").

7.2 mlpack executable versions

Each mlpack executable supports the `-version` (or `-V`) option, which will print the version of mlpack used. If the version is not an official release but instead from svn trunk, the version will be "mlpack trunk" (and may have a revision number appended to "trunk").

Chapter 8

Density Estimation Tree (DET) tutorial

8.1 Introduction

DETs perform the unsupervised task of density estimation using decision trees. Using a trained density estimation tree (DET), the density at any particular point can be estimated very quickly (O(log n) time, where n is the number of points the tree is built on).

The details of this work is presented in the following paper:

```
@inproceedings{ram2011density,
  title={Density estimation trees},
  author={Ram, P. and Gray, A.G.},
  booktitle={Proceedings of the 17th ACM SIGKDD International Conference on
      Knowledge Discovery and Data Mining},
  pages={627--635},
  year={2011},
  organization={ACM}
}
```

mlpack provides:

- a **simple command-line executable** (p. 22) to perform density estimation and related analyses using DETs
- a **generic C++ class (DTree)** (p. 24) which provides various functionality for the DETs
- a set of functions in the namespace **mlpack::det** (p. 25) to perform cross-validation for the task of density estimation with DETs

8.2 Table of Contents

A list of all the sections this tutorial contains.

8.3 Command-Line 'det'

The command line arguments of this program can be viewed using the '-h' option:

```
$ det -h
Density Estimation With Density Estimation Trees

  This program performs a number of functions related to Density Estimation
  Trees.  The optimal Density Estimation Tree (DET) can be trained on a set of
  data (specified by --train_file) using cross-validation (with number of folds
  specified by --folds).  In addition, the density of a set of test points
  (specified by --test_file) can be estimated, and the importance of each
  dimension can be computed.  If class labels are given for the training points
  (with --labels_file), the class memberships of each leaf in the DET can be
  calculated.

  The created DET can be saved to a file, along with the density estimates for
  the test set and the variable importances.

Required options:

  --train_file (-t) [string]    The data set on which to build a density
                                estimation tree.

Options:

  --folds (-f) [int]            The number of folds of cross-validation to
                                perform for the estimation (0 is LOOCV)  Default
                                value 10.
  --help (-h)                   Default help info.
  --info [string]               Get help on a specific module or option.
                                Default value ''.
  --labels_file (-l) [string]   The labels for the given training data to
                                generate the class membership of each leaf (as
                                an extra statistic)  Default value ''.
  --leaf_class_table_file (-L) [string]
                                The file in which to output the leaf class
                                membership table.  Default value
                                'leaf_class_membership.txt'.
  --max_leaf_size (-M) [int]    The maximum size of a leaf in the unpruned,
                                fully grown DET.  Default value 10.
  --min_leaf_size (-N) [int]    The minimum size of a leaf in the unpruned,
                                fully grown DET.  Default value 5.
  --print_tree (-p)             Print the tree out on the command line (or in
                                the file specified with --tree_file).
  --print_vi (-I)               Print the variable importance of each feature
                                out on the command line (or in the file
                                specified with --vi_file).
  --test_file (-T) [string]     A set of test points to estimate the density of.
                                 Default value ''.
  --test_set_estimates_file (-E) [string]
                                The file in which to output the estimates on the
                                test set from the final optimally pruned tree.
                                Default value ''.
  --training_set_estimates_file (-e) [string]
                                The file in which to output the density
                                estimates on the training set from the final
```

```
                                optimally pruned tree.  Default value ''.
  --tree_file (-r) [string]     The file in which to print the final optimally
                                pruned tree.  Default value ''.
  --unpruned_tree_estimates_file (-u) [string]
                                The file in which to output the density
                                estimates on the training set from the large
                                unpruned tree.  Default value ''.
  --verbose (-v)                Display informational messages and the full list
                                of parameters and timers at the end of
                                execution.
  --vi_file (-i) [string]       The file to output the variable importance
                                values for each feature.  Default value ''.

For further information, including relevant papers, citations, and theory,
consult the documentation found at http://www.mlpack.org or included with your
distribution of MLPACK.
```

8.3.1 Plain-vanilla density estimation

We can just train a DET on the provided data set *S*. Like all datasets **mlpack** uses, the data should be row-major (**mlpack** transposes data when it is loaded; internally, the data is column-major – see **this page** (p. 13) for more information).

```
$ det -t dataset.csv -v
```

By default, det performs 10-fold cross-validation (using the α-pruning regularization for decision trees). To perform LOOCV (leave-one-out cross-validation), which can provide better results but will take longer, use the following command:

```
$ det -t dataset.csv -f 0 -v
```

To perform k-fold crossvalidation, use `-f k` (or `-folds k`). There are certain other options available for training. For example, in the construction of the initial tree, you can specify the maximum and minimum leaf sizes. By default, they are 10 and 5 respectively; you can set them using the `-M` (`-max_leaf_size`) and the `-N` (`-min_leaf_size`) options.

```
$ det -t dataset.csv -M 20 -N 10
```

In case you want to output the density estimates at the points in the training set, use the `-e` (`-training_set_estimates_file`) option to specify the output file to which the estimates will be saved. The first line in density_estimates.txt will correspond to the density at the first point in the training set. Note that the logarithm of the density estimates are given, which allows smaller estimates to be saved.

```
$ det -t dataset.csv -e density_estimates.txt -v
```

8.3.2 Estimation on a test set

Often, it is useful to train a density estimation tree on a training set and then obtain density estimates from the learned estimator for a separate set of test points. The `-T` (`-test_file`) option allows specification of a set of test points, and the `-E` (`-test_set_estimates_file`) option allows specification of the file into which the test set estimates are saved. Note that the logarithm of the density estimates are saved; this allows smaller values to be saved.

```
$ det -t dataset.csv -T test_points.csv -E test_density_estimates.txt -v
```

8.3.3 Printing a trained DET

A depth-first visualization of the DET can be obtained by using the `-p` (`-print_tree`) flag.

```
$ det -t dataset.csv -p -v
```

To print this tree in a file, use the `-r` (`--tree_file`) option to specify the output file along with the `-P` (`--print_tree`) flag.

```
$ det -t dataset.csv -p -r tree.txt -v
```

8.3.4 Computing the variable importance

The variable importance (with respect to density estimation) of the different features in the data set can be obtained by using the `-I` (`--print_vi`) option. This outputs the absolute (as opposed to relative) variable importance of the all the features.

```
$ det -t dataset.csv -I -v
```

To print this in a file, use the `-i` (`--vi_file`) option.

```
$ det -t dataset.csv -I -i variable_importance.txt -v
```

8.3.5 Leaf Membership

In case the dataset is labeled and you want to find the class membership of the leaves of the tree, there is an option to print the class membership into a file. The training data has to still be input in an unlabeled format, but an additional label file containing the corresponding labels of each point has to be input using the `-l` (`--labels_file`) option. The file to output the class memberships into can be specified with `-L` (`--leaf_class_table_file`). If `-L` is left unspecified, leaf_class_membership.txt is used by default.

```
$ det -t dataset.csv -l labels.csv -v
$ det -t dataset.csv -l labels.csv -L leaf_class_membership_file.txt -v
```

8.4 The 'DTree' class

This class implements density estimation trees. Below is a simple example which initializes a density estimation tree.

```
#include <mlpack/methods/det/dtree.hpp>

using namespace mlpack::det;

// The dataset matrix, on which to learn the density estimation tree.
extern arma::Mat<float> data;

// Initialize the tree.  This function also creates and saves the bounding box
// of the data.  Note that it does not actually build the tree.
DTree<> det(data);
```

8.4.1 Public Functions

The function `Grow()` greedily grows the tree, adding new points to the tree. Note that the points in the dataset will be reordered. This should only be run on a tree which has not already been built. In general, it is more useful to use the **`Trainer()`** (p. 88) function found in **'namespace mlpack::det'** (p. 25).

```
// This keeps track of the data during the shuffle that occurs while growing the
// tree.
arma::Col<size_t> oldFromNew(data.n_cols);
for (size_t i = 0; i < data.n_cols; i++)
  oldFromNew[i] = i;

// This function grows the tree down to the leaves. It returns the current
// minimum value of the regularization parameter alpha.
size_t maxLeafSize = 10;
size_t minLeafSize = 5;

double alpha = det.Grow(data, oldFromNew, false, maxLeafSize, minLeafSize);
```

Note that the alternate volume regularization should not be used (see ticket #238).

To estimate the density at a given query point, use the following code. Note that the logarithm of the density is returned.

```
// For a given query, you can obtain the density estimate.
extern arma::Col<float> query;
extern DTree* det;
double estimate = det->ComputeValue(&query);
```

Computing the **variable importance** of each feature for the given DET.

```
// The data matrix and density estimation tree.
extern arma::mat data;
extern DTree* det;

// The variable importances will be saved into this vector.
arma::Col<double> varImps;

// You can obtain the variable importance from the current tree.
det->ComputeVariableImportance(varImps);
```

8.5 'namespace mlpack::det'

The functions in this namespace allows the user to perform tasks with the 'DTree' class. Most importantly, the **`Trainer()`** (p. 88) method allows the full training of a density estimation tree with cross-validation. There are also utility functions which allow printing of leaf membership and variable importance.

8.5.1 Utility Functions

The code below details how to train a density estimation tree with cross-validation.

```
#include <mlpack/methods/det/dt_utils.hpp>

using namespace mlpack::det;

// The dataset matrix, on which to learn the density estimation tree.
extern arma::Mat<float> data;

// The number of folds for cross-validation.
const size_t folds = 10; // Set folds = 0 for LOOCV.

const size_t maxLeafSize = 10;
const size_t minLeafSize = 5;

// Train the density estimation tree with cross-validation.
DTree<>* dtree_opt = Trainer(data, folds, false, maxLeafSize, minLeafSize);
```

Note that the alternate volume regularization should be set to false because it has known bugs (see #238).

To print the class membership of leaves in the tree into a file, see the following code.

```
extern arma::Mat<size_t> labels;
extern DTree* det;
const size_t numClasses = 3; // The number of classes must be known.

extern string leafClassMembershipFile;

PrintLeafMembership(det, data, labels, numClasses, leafClassMembershipFile);
```

Note that you can find the number of classes with `max(labels) + 1`. The variable importance can also be printed to a file in a similar manner.

```
extern DTree* det;

extern string variableImportanceFile;
const size_t numFeatures = data.n_rows;

PrintVariableImportance(det, numFeatures, variableImportanceFile);
```

8.6 Further Documentation

For further documentation on the DTree class, consult the **complete API documentation** (p. 145).

Chapter 9

EMST Tutorial

9.1 Introduction

The Euclidean Minimum Spanning Tree problem is widely used in machine learning and data mining applications. Given a set S of points in $\mathbf{R}^d$, our task is to compute lowest weight spanning tree in the complete graph on S with edge weights given by the Euclidean distance between points.

Among other applications, the EMST can be used to compute hierarchical clusterings of data. A *single-linkage clustering* can be obtained from the EMST by deleting all edges longer than a given cluster length. This technique is also referred to as a *Friends-of-Friends* clustering in the astronomy literature.

MLPACK includes an implementation of **Dual-Tree Boruvka** on kd-trees, the empirically and theoretically fastest EMST algorithm. For more details, see March, *et al.*, *Euclidean Minimum Spanning Tree: Algorithm, Analysis, and Applications*, in KDD, 2010. An implementation using cover trees is forthcoming.

mlpack provides:

- a **simple command-line executable** (p. 27) to compute the EMST of a given data set
- a **simple C++ interface** (p. 28) to compute the EMST

9.2 Table of Contents

A list of all the sections this tutorial contains.

9.3 Command-Line 'EMST'

The emst executable in **mlpack** will compute the EMST of a given set of points and store the resulting edge list to a file.

The output file contains an edge list representation of the MST in an $n-1 \times 3$ matrix, where the first and second columns are labels of points and the third column is the edge weight. The edges are sorted in order of increasing weight.

Below are several examples of simple usage (and the resultant output). The '-v' option is used so that verbose output is given. Further documentation on each individual option can be found by typing

```
$ emst --help
```

```
$ emst --input_file=dataset.csv --output_file=edge_list.csv -v
[INFO ] Reading in data.
[INFO ] Loading 'dataset.csv' as CSV data.
[INFO ] Data read, building tree.
[INFO ] Tree built, running algorithm.
[INFO ] 4 edges found so far.
[INFO ] 5 edges found so far.
[INFO ] Total spanning tree length: 1002.45
[INFO ] Saving CSV data to 'edge_list.csv'.
[INFO ]
[INFO ] Execution parameters:
[INFO ]   help: false
[INFO ]   info: ""
[INFO ]   input_file: dataset.csv
[INFO ]   leaf_size: 1
[INFO ]   naive: false
[INFO ]   output_file: edge_list.csv
[INFO ]   verbose: true
[INFO ]
[INFO ] Program timers:
[INFO ]   emst/mst_computation: 0.000179s
[INFO ]   emst/tree_building: 0.000061s
[INFO ]   total_time: 0.052641s
```

The code performs at most $\log N$ iterations for N data points. It will print an update on the number of MST edges found after each iteration. Convenient program timers are given for different parts of the calculation at the bottom of the output, as well as the parameters the simulation was run with.

```
$ cat dataset.csv
0, 0
1, 1
3, 3
0.5, 0
1000, 0
1001, 0

$ cat edge_list.csv
0.0000000000e+00,3.0000000000e+00,5.0000000000e-01
4.0000000000e+00,5.0000000000e+00,1.0000000000e+00
1.0000000000e+00,3.0000000000e+00,1.1180339887e+00
1.0000000000e+00,2.0000000000e+00,2.8284271247e+00
2.0000000000e+00,4.0000000000e+00,9.9700451353e+02
```

The input points are labeled 0-5. The output tells us that the MST connects point 0 to point 3, point 4 to point 5, point 1 to point 3, point 1 to point 2, and point 2 to point 4, with the corresponding edge weights given in the third column. The total length of the MST is also given in the verbose output.

Note that it is also possible to compute the EMST using a naive ($O(N^2)$) algorithm for timing and comparison purposes.

9.4 The 'DualTreeBoruvka' class

The 'DualTreeBoruvka' class contains our implementation of the Dual-Tree Boruvka algorithm.

The class has two constructors: the first takes the data set, constructs the kd-tree, and computes the MST. The second takes data set and an already constructed tree.

The class provides one method that performs the MST computation:

```
void ComputeMST(const arma::mat& results);
```

This method stores the computed MST in the matrix results in the format given above.

9.5 Further documentation

For further documentation on the DualTreeBoruvka class, consult the **complete API documentation** (p. 172).

Chapter 10

Fast max-kernel search tutorial (fastmks)

10.1 Introduction

The FastMKS algorithm (fast exact max-kernel search) is a recent algorithm proposed in the following paper:

```
@inproceedings{curtin2013fast,
  title={Fast Exact Max-Kernel Search},
  author={Curtin, Ryan R. and Ram, Parikshit and Gray, Alexander G.},
  booktitle={Proceedings of the 2013 SIAM International Conference on Data
      Mining (SDM '13)},
  year={2013},
  pages={1--9}
}
```

Given a set of query points Q and a set of reference points R, the FastMKS algorithm is a fast dual-tree (or single-tree) algorithm which finds

$$\arg\max_{p_r \in R} K(p_q, p_r)$$

for all points $p_q \in Q$ and for some Mercer kernel $K(\cdot, \cdot)$. A Mercer kernel is a kernel that is positive semidefinite; these are the classes of kernels that can be used with the kernel trick. In short, the positive semidefiniteness of a Mercer kernel means that any kernel matrix (or Gram matrix) created on a dataset must be positive semidefinite.

The FastMKS algorithm builds trees on the datasets Q and R in such a way that explicit representation of the points in the kernel space is unnecessary, by using cover trees (**mlpack::tree::CoverTree** (p. 491)). This allows the algorithm to be run, for instance, on string kernels, where there is no sensible explicit representation. The **mlpack** implementation allows any type of tree that does not require an explicit representation to be used. For more details, see the paper.

At the time of this writing there is no other fast algorithm for exact max-kernel search. Also, **mlpack** implements dual-tree FastMKS, while the paper referenced above only explains single-tree search.

mlpack provides:

- a **simple command-line executable** (p. 32) to run FastMKS
- a **C++ interface** (p. 34) to run FastMKS

10.2 Table of Contents

A list of all the sections this tutorial contains.

10.3 Command-line FastMKS (fastmks)

mlpack provides a command-line program, `fastmks`, which is used to perform FastMKS on a given query and reference dataset. It supports numerous different types of kernels:

- **linear kernel** (p. 248)
- **polynomial kernel** (p. 249)
- **cosine distance** (p. 229)
- **Gaussian kernel** (p. 235)
- **Epanechnikov kernel** (p. 230)
- **triangular kernel** (p. 257)
- **hyperbolic tangent kernel** (p. 239)
- **Laplacian kernel** (p. 246)

Note that when a shift-invariant kernel is used, the results will be the same as nearest neighbor search, so **allknn** (p. 55) may be a better option. A shift-invariant kernel is a kernel that depends only on the distance between the two input points. The **Gaussian kernel** (p. 235), **Epanechnikov kernel** (p. 230), **triangular kernel** (p. 257), and **Laplacian kernel** (p. 246) are instances of shift-invariant kernels. The paper contains more details on this situation. The `fastmks` executable still provides these kernels as options, though.

The following examples detail usage of the `fastmks` program. Note that you can get documentation on all the possible parameters by typing:

```
$ fastmks --help
```

10.3.1 FastMKS with a linear kernel on one dataset

If only one dataset is specified (with `-r` or `--reference_file`), the reference dataset is taken to be both the query and reference datasets. The example below finds the 4 maximum kernels of each point in dataset.csv, using the default linear kernel.

```
$ fastmks -r dataset.csv -k 4 -v -p products.csv -i indices.csv
```

When the operation completes, the values of the kernels are saved in products.csv and the indices of the points which give the maximum kernels are saved in indices.csv.

```
$ head indices.csv
762,910,863,890
762,910,426,568
910,762,863,426
762,910,863,426
863,910,614,762
762,863,910,614
762,910,488,568
762,910,863,426
910,762,863,426
863,762,910,614
```

```
$ head products.csv
1.6221652894e+00,1.5998743443e+00,1.5898890769e+00,1.5406789753e+00
1.3387953449e+00,1.3317349486e+00,1.2966613184e+00,1.2774493620e+00
1.6386110476e+00,1.6332029753e+00,1.5952629124e+00,1.5887195330e+00
1.0917545803e+00,1.0820878726e+00,1.0668992636e+00,1.0419838050e+00
1.2272441028e+00,1.2169643942e+00,1.2104597963e+00,1.2067780154e+00
1.5720962456e+00,1.5618504956e+00,1.5609069923e+00,1.5235605095e+00
1.3655478674e+00,1.3548593212e+00,1.3311547298e+00,1.3250728881e+00
2.0119149744e+00,2.0043668067e+00,1.9847289214e+00,1.9298280046e+00
1.1586923205e+00,1.1494586097e+00,1.1274872962e+00,1.1248172766e+00
4.4789820372e-01,4.4618539778e-01,4.4200024852e-01,4.3989721792e-01
```

We can see in this example that for point 0, the point with maximum kernel value is point 762, with a kernel value of 1.622165. For point 3, the point with third largest kernel value is point 863, with a kernel value of 1.0669.

10.3.2 FastMKS on a reference and query dataset

The query points may be different than the reference points. To specify a different query set, the `-q` (or `--query_file`) option is used, as in the example below.

```
$ fastmks -q query_set.csv -r reference_set.csv -k 5 -i indices.csv -p products.csv
```

10.3.3 FastMKS with a different kernel

The `fastmks` program offers more than just the linear kernel. Valid options are `'linear'`, `'polynomial'`, `'cosine'`, `'gaussian'`, `'epanechnikov'`, `'triangular'`, `'laplacian'`, and `'hyptan'` (the hyperbolic tangent kernel). Note that the hyperbolic tangent kernel is provably not a Mercer kernel but is positive semidefinite on most datasets and is commonly used as a kernel. Note also that the Gaussian kernel and other shift-invariant kernels give the same results as nearest neighbor search (see **NeighborSearch tutorial (k-nearest-neighbors)** (p. 55)).

The kernel to use is specified with the `-K` (or `--kernel`) option. The example below uses the cosine similarity as a kernel.

```
$ fastmks -r dataset.csv -k 5 -K cosine -i indices.csv -p products.csv -v
```

10.3.4 Using single-tree search or naive search

In some cases, it may be useful to not use the dual-tree FastMKS algorithm. Instead you can specify the `-single` option, indicating that a tree should be built only on the reference set, and then the queries should be processed in a linear scan (instead of in a tree). Alternately, the `-N` (or `-naive`) option makes the program not build trees at all and instead use brute-force search to find the solutions.

The example below uses single-tree search on two datasets.

```
$ fastmks -q query_set.csv -r reference_set.csv --single -k 5 -p products.csv \
> -i indices.csv
```

The example below uses naive search on one dataset.

```
$ fastmks -r reference_set.csv -k 5 -N -p products.csv -i indices.csv
```

10.3.5 Paramters for alternate kernels

Many of the alternate kernel choices have parameters which can be chosen; these are detailed in this section.

- **`-w`** (`-bandwidth`): this sets the bandwidth of the kernel, and is applicable to the `'gaussian'`, `'epanechnikov'`, and `'triangular'` kernels. This is the "spread" of the kernel.
- **`-d`** (`-degree`): this sets the degree of the polynomial kernel (the power to which the result is raised). It is only applicable to the `'polynomial'` kernel.
- **`-o`** (`-offset`): this sets the offset of the kernel, for the `'polynomial'` and `'hyptan'` kernel. See **the polynomial kernel documentation** (p. 249) and **the hyperbolic tangent kernel documentation** (p. 239) for more information.
- **`-s`** (`-scale`): this sets the scale of the kernel, and is only applicable to the `'hyptan'` kernel. See **the hyperbolic tangent kernel documentation** (p. 239) for more information.

10.4 The 'FastMKS' class

The `FastMKS<>` class offers a simple API for use within C++ applications, and allows further flexibility in kernel choice and tree type choice. However, `FastMKS<>` has no default template parameter for the kernel type – that must be manually specified. Choices that **mlpack** provides include:

- **mlpack::kernel::LinearKernel** (p. 248)
- **mlpack::kernel::PolynomialKernel** (p. 249)
- **mlpack::kernel::CosineDistance** (p. 229)
- **mlpack::kernel::GaussianKernel** (p. 235)
- **mlpack::kernel::EpanechnikovKernel** (p. 230)
- **mlpack::kernel::TriangularKernel** (p. 257)
- **mlpack::kernel::HyperbolicTangentKernel** (p. 239)
- **mlpack::kernel::LaplacianKernel** (p. 246)
- **mlpack::kernel::PSpectrumStringKernel** (p. 252)

The following examples use kernels from that list. Writing your own kernel is detailed in **the next section** (p. 37). Remember that when you are using the C++ interface, the data matrices must be column-major. See **Matrices in MLPACK** (p. 13) for more information.

10.4.1 FastMKS on one dataset

Given only a reference dataset, the following code will run FastMKS with k set to 5.

```
#include <mlpack/methods/fastmks/fastmks.hpp>
#include <mlpack/core/kernels/linear_kernel.hpp>

using namespace mlpack::fastmks;

// The reference dataset, which is column-major.
extern arma::mat data;

// This will initialize the FastMKS object with the linear kernel with default
// options: K(x, y) = x^T y.  The tree is built in the constructor.
FastMKS<LinearKernel> f(data);

// The results will be stored in these matrices.
arma::Mat<size_t> indices;
arma::mat products;

// Run FastMKS.
f.Search(5, indices, products);
```

10.4.2 FastMKS with a query and reference dataset

In this setting we have both a query and reference dataset. We search for 10 maximum kernels.

```
#include <mlpack/methods/fastmks/fastmks.hpp>
#include <mlpack/core/kernels/triangular_kernel.hpp>

using namespace mlpack::fastmks;
using namespace mlpack::kernel;

// The reference and query datasets, which are column-major.
extern arma::mat referenceData;
extern arma::mat queryData;

// This will initialize the FastMKS object with the triangular kernel with
// default options (bandwidth of 1).  The trees are built in the constructor.
FastMKS<TriangularKernel> f(queryData, referenceData);

// The results will be stored in these matrices.
arma::Mat<size_t> indices;
arma::mat products;

// Run FastMKS.
f.Search(10, indices, products);
```

10.4.3 FastMKS with an initialized kernel

Often, kernels have parameters which need to be specified. `FastMKS<>` has constructors which take initialized kernels. Note that temporary kernels cannot be passed as an argument. The example below initializes a `PolynomialKernel` object and then runs FastMKS with a query and reference dataset.

```
#include <mlpack/methods/fastmks/fastmks.hpp>
#include <mlpack/core/kernels/polynomial_kernel.hpp>

using namespace mlpack::fastmks;
using namespace mlpack::kernel;

// The reference and query datasets, which are column-major.
extern arma::mat referenceData;
```

```
extern arma::mat queryData;

// Initialize the polynomial kernel with degree of 3 and offset of 2.5.
PolynomialKernel pk(3.0, 2.5);

// Create the FastMKS object with the initialized kernel.
FastMKS<PolynomialKernel> f(referenceData, queryData, pk);

// The results will be stored in these matrices.
arma::Mat<size_t> indices;
arma::mat products;

// Run FastMKS.
f.Search(10, indices, products);
```

The syntax for running FastMKS with one dataset and an initialized kernel is very similar:

```
FastMKS<PolynomialKernel> f(referenceData, pk);
```

10.4.4 FastMKS with an already-created tree

By default, `FastMKS<>` uses the cover tree datastructure (see **mlpack::tree::CoverTree** (p. 491)). Sometimes, it is useful to modify the parameters of the cover tree. In this scenario, a tree must be built outside of the constructor, and then passed to the appropriate `FastMKS<>` constructor. An example on just a reference dataset is shown below, where the base of the cover tree is modified.

We also use an instantiated kernel, but because we are building our own tree, we must use **IPMetric** (p. 286) so that our tree is built on the metric induced by our kernel function.

```
#include <mlpack/methods/fastmks/fastmks.hpp>
#include <mlpack/core/kernels/polynomial_kernel.hpp>

// The reference dataset, which is column-major.
extern arma::mat data;

// Initialize the polynomial kernel with a degree of 4 and offset of 2.0.
PolynomialKernel pk(4.0, 2.0);

// Create the metric induced by this kernel (because a kernel is not a metric
// and we can't build a tree on a kernel alone).
IPMetric<PolynomialKernel> metric(pk);

// Now build a tree on the reference dataset using the instantiated metric and
// the custom base of 1.5 (default is 1.3).  We have to be sure to use the right
// type here -- FastMKS needs the FastMKSStat object as the tree's
// StatisticType.
typedef tree::CoverTree<IPMetric<PolynomialKernel>, tree::FirstPointIsRoot,
    FastMKSStat> TreeType; // Convenience typedef.
TreeType* tree = new TreeType(data, metric, 1.5);

// Now initialize FastMKS with that statistic.  We don't need to specify the
// TreeType template parameter since we are still using the default.  We don't
// need to pass the kernel because that is contained in the tree.
FastMKS<PolynomialKernel> f(data, tree);

// The results will be stored in these matrices.
arma::Mat<size_t> indices;
arma::mat products;

// Run FastMKS.
f.Search(10, indices, products);
```

The syntax is similar for the case where different query and reference datasets are given; but trees for both need to be built in the manner specified above. Be sure to build both trees using the same metric (or at least a metric with the exact same parameters).

```
FastMKS<PolynomialKernel> f(referenceData, referenceTree, queryData, queryTree);
```

10.5 Writing a custom kernel for FastMKS

While **mlpack** provides some number of kernels in the **mlpack::kernel** (p. 93) namespace, it is easy to create a custom kernel. To satisfy the KernelType policy, a class must implement the following methods:

```
// Empty constructor is required.
KernelType();

// Evaluate the kernel between two points.
template<typename VecType>
double Evaluate(const VecType& a, const VecType& b);
```

The template parameter `VecType` is helpful (but not necessary) so that the kernel can be used with both sparse and dense matrices (`arma::sp_mat` and `arma::mat`).

10.6 Using other tree types for FastMKS

The use of the cover tree (see **CoverTree** (p. 491)) is not necessary for FastMKS, although it is the default tree type. A different type of tree can be specified with the TreeType template parameter. However, the tree type is required to have **FastMKSStat** (p. 197) as the StatisticType, and for FastMKS to work, the tree must be built only on kernel evaluations (or distance evaluations in the kernel space via **IPMetric::Evaluate()** (p. 286)).

Below is an example where a custom tree class, `CustomTree`, is used as the tree type for FastMKS. In this example FastMKS is only run on one dataset.

```
#include <mlpack/methods/fastmks/fastmks.hpp>
#include "custom_tree.hpp"

using namespace mlpack::fastmks;
using namespace mlpack::tree;

// The dataset that FastMKS will be run on.
extern arma::mat data;

// The custom tree type.  We'll assume that the first template parameter is the
// statistic type.
typedef CustomTree<FastMKSStat> TreeType;

// The FastMKS constructor will create the tree.
FastMKS<LinearKernel, TreeType> f(data);

// These will hold the results.
arma::Mat<size_t> indices;
arma::mat products;

// Run FastMKS.
f.Search(5, indices, products);
```

10.7 Running FastMKS on objects

FastMKS has a lot of utility on objects which are not representable in some sort of metric space. These objects might be strings, graphs, models, or other objects. For these types of objects, questions based on distance don't really make sense. One good example is with strings. The question "how far is 'dog' from 'Taki Inoue'?" simply doesn't make sense. We can't have a centroid of the terms 'Fritz', 'E28', and 'popsicle'.

However, what we can do is define some sort of kernel on these objects. These kernels generally correspond to some similarity measure, with one example being the p-spectrum string kernel (see **mlpack::kernel::PSpectrumStringKernel** (p. 252)). Using that, we can say "how similar is 'dog' to 'Taki Inoue'?" and get an actual numerical result by evaluating K('dog', 'Taki Inoue') (where K is our p-spectrum string kernel).

The only requirement on these kernels is that they are positive definite kernels (or Mercer kernels). For more information on those details, refer to the FastMKS paper.

Remember that FastMKS is a tree-based method. But trees like the binary space tree require centroids – and as we said earlier, centroids often don't make sense with these types of objects. Therefore, we need a type of tree which is built **exclusively** on points in the dataset – those are points which we can evaluate our kernel function on. The cover tree is one example of a type of tree satisfying this condition; its construction will only call the kernel function on two points that are in the dataset.

But, we have one more problem. The `CoverTree` class is built on `arma::mat` objects (dense matrices). Our objects, however, are not necessarily representable in a column of a matrix. To use the example we have been using, strings cannot be represented easily in a matrix because they may all have different lengths.

The way to work around this problem is to create a "fake" data matrix which simply holds indices to objects. A good example of how to do this is detailed in the documentation for the **PSpectrumStringKernel** (p. 252).

In short, the trick is to make each data matrix one-dimensional and containing linear indices:

```
arma::mat data = "0 1 2 3 4 5 6 7 8";
```

Then, when `Evaluate()` is called on the kernel function, the parameters will be two one-dimensional vectors that simply contain indices to objects. The example below details the process a little better:

```
// This function evaluates the kernel on two Objects (in this example, its
// implementation is not important; the only important thing is that the
// function exists).
double ObjectKernel::Evaluate(const Object& a, const Object& b) const;

template<typename VecType>
double ObjectKernel::Evaluate(const VecType& a, const VecType& b) const
{
  // Extract the indices from the vectors.
  const size_t indexA = size_t(a[0]);
  const size_t indexB = size_t(b[0]);

  // Assume that 'objects' is an array (or std::vector or other container)
  // holding Objects.
  const Object& objectA = objects[indexA];
  const Object& objectB = objects[indexB];

  // Now call the function that does the actual evaluation on the objects and
  // return its result.
  return Evaluate(objectA, objectB);
}
```

As written earlier, the documentation for **PSpectrumStringKernel** (p. 252) is a good place to consult for further reference on this. That kernel uses two dimensional indices; one dimension represents the index of the string, and the other represents whether it is referring to the query set or the reference set. If your kernel is meant to work on separate query and reference sets, that strategy should be considered.

10.8 Further documentation

For further documentation on the FastMKS class, consult the **complete API documentation** (p. 185).

Chapter 11

K-Means tutorial (kmeans)

11.1 Introduction

The popular k-means algorithm for clustering has been around since the late 1950s, and the standard algorithm was proposed by Stuart Lloyd in 1957. Given a set of points X, k-means clustering aims to partition each point x_i into a cluster c_j (where $j \leq k$ and k, the number of clusters, is a parameter). The partitioning is done to minimize the objective function

$$\sum_{j=1}^{k} \sum_{x_i \in c_j} \|x_i - \mu_j\|^2$$

where μ_j is the centroid of cluster c_j. The standard algorithm is a two-step algorithm:

- **Assignment step**. Each point x_i in X is assigned to the cluster whose centroid it is closest to.
- **Update step**. Using the new cluster assignments, the centroids of each cluster are recalculated.

The algorithm has converged when no more assignment changes are happening with each iteration. However, this algorithm can get stuck in local minima of the objective function and is particularly sensitive to the initial cluster assignments. Also, situations can arise where the algorithm will never converge but reaches steady state – for instance, one point may be changing between two cluster assignments.

There is vast literature on the k-means algorithm and its uses, as well as strategies for choosing initial points effectively and keeping the algorithm from converging in local minima. **mlpack** does implement some of these, notably the Bradley-Fayyad algorithm (see the reference below) for choosing refined initial points. Importantly, the C++ `KMeans` class makes it very easy to improve the k-means algorithm in a modular way.

```
@inproceedings{bradley1998refining,
  title={Refining initial points for k-means clustering},
  author={Bradley, Paul S. and Fayyad, Usama M.},
  booktitle={Proceedings of the Fifteenth International Conference on Machine
      Learning (ICML 1998)},
  volume={66},
  year={1998}
}
```

mlpack provides:

- a **simple command-line executable** (p. 40) to run k-means
- a **simple C++ interface** (p. 42) to run k-means
- a **generic, extensible, and powerful C++ class** (p. 45) for complex usage

11.2 Table of Contents

A list of all the sections this tutorial contains.

11.3 Command-Line 'kmeans'

mlpack provides a command-line executable, `kmeans`, to allow easy execution of the k-means algorithm on data. Complete documentation of the executable can be found by typing

```
$ kmeans --help
```

Below are several examples demonstrating simple use of the `kmeans` executable.

11.3.1 Simple k-means clustering

We want to find 5 clusters using the points in the file dataset.csv. By default, if any of the clusters end up empty, that cluster will be reinitialized to contain the point furthest from the cluster with maximum variance. The cluster assignments of each point will be stored in assignments.csv. Each row in assignments.csv will correspond to the row in dataset.csv.

```
$ kmeans -c 5 -i dataset.csv -v -o assignments.csv
```

11.3.2 Saving the resulting centroids

Sometimes it is useful to save the centroids of the clusters found by k-means; one example might be for plotting the points. The `-C` (`--centroid_file`) option allows specification of a file into which the centroids will be saved (one centroid per line, if it is a CSV or other text format).

```
$ kmeans -c 5 -i dataset.csv -v -o assignments.csv -C centroids.csv
```

11.3.3 Allowing empty clusters

If you would like to allow empty clusters to exist, instead of reinitializing them, simply specify the `-e` (`--allow_empty_clusters`) option. Note that when you save your clusters, some of the clusters may be filled with NaNs. This is expected behavior – if a cluster has no points, the concept of a centroid makes no sense.

```
$ kmeans -c 5 -i dataset.csv -v -o assignments.csv -C centroids.csv
```

11.3.4 Limiting the maximum number of iterations

As mentioned earlier, the k-means algorithm can often fail to converge. In such a situation, it may be useful to stop the algorithm by way of limiting the maximum number of iterations. This can be done with the `-m` (`--max_iterations`) parameter, which is set to 1000 by default. If the maximum number of iterations is 0, the algorithm will run until convergence – or potentially forever. The example below sets a maximum of 250 iterations.

```
$ kmeans -c 5 -i dataset.csv -v -o assignments.csv -m 250
```

11.3.5 Setting the overclustering factor

The **mlpack** k-means implementation allows "overclustering", which is when the k-means algorithm is run with more than the requested number of clusters. Upon convergence, the clusters with the nearest centroids are merged until only the requested number of centroids remain. This can provide better clustering results. The overclustering factor, specified with `-O` or `--overclustering`, determines how many more clusters are found than were requested. For instance, with `k` set to 5 and an overclustering factor of 2, 10 clusters will be found. Note that the overclustering factor does not need to be an integer.

The following code snippet finds 5 clusters, but with an overclustering factor of 2.4 (so 12 clusters are found and then merged together to produce 5 final clusters).

```
$ kmeans -c 5 -O 2.4 -i dataset.csv -v -o assignments.csv
```

11.3.6 Using Bradley-Fayyad "refined start"

The method proposed by Bradley and Fayyad in their paper "Refining initial points for k-means clustering" is implemented in **mlpack**. This strategy samples points from the dataset and runs k-means clustering on those points multiple times, saving the resulting clusters. Then, k-means clustering is run on those clusters, yielding the original number of clusters. The centroids of those resulting clusters are used as initial centroids for k-means clustering on the entire dataset.

This technique generally gives better initial points than the default random partitioning, but depending on the parameters, it can take much longer. This initialization technique is enabled with the `-r` (`--refined_start`) option. The `-S` (`--samplings`) parameter controls how many samplings of the dataset are performed, and the `-p` (`--percentage`) parameter controls how much of the dataset is randomly sampled for each sampling (it must be between 0.0 and 1.0). For more information on the refined start technique, see the paper referenced in the introduction of this tutorial.

The example below performs k-means clustering, giving 5 clusters, using the refined start technique, sampling 10% of the dataset 25 times to produce the initial centroids.

```
$ kmeans -c 5 -i dataset.csv -v -o assignments.csv -r -S 25 -p 0.2
```

11.4 The 'KMeans' class

The `KMeans<>` class (with default template parameters) provides a simple way to run k-means clustering using **mlpack** in C++. The default template parameters for `KMeans<>` will initialize cluster assignments randomly and disallow empty clusters. When an empty cluster is encountered, the point furthest from the cluster with maximum variance is set to the centroid of the empty cluster.

11.4.1 Running k-means and getting cluster assignments

The simplest way to use the `KMeans<>` class is to pass in a dataset and a number of clusters, and receive the cluster assignments in return. Note that the dataset must be column-major – that is, one column corresponds to one point. See **the matrices guide** (p. 13) for more information.

```
#include <mlpack/methods/kmeans/kmeans.hpp>

using namespace mlpack::kmeans;

// The dataset we are clustering.
extern arma::mat data;
// The number of clusters we are getting.
extern size_t clusters;

// The assignments will be stored in this vector.
arma::Col<size_t> assignments;

// Initialize with the default arguments.
KMeans<> k;
k.Cluster(data, clusters, assignments);
```

Now, the vector `assignments` holds the cluster assignments of each point in the dataset.

11.4.2 Running k-means and getting centroids of clusters

Often it is useful to not only have the cluster assignments, but the centroids of each cluster. Another overload of `Cluster()` makes this easily possible:

```
#include <mlpack/methods/kmeans/kmeans.hpp>

using namespace mlpack::kmeans;

// The dataset we are clustering.
extern arma::mat data;
// The number of clusters we are getting.
extern size_t clusters;

// The assignments will be stored in this vector.
arma::Col<size_t> assignments;
// The centroids will be stored in this matrix.
arma::mat centroids;

// Initialize with the default arguments.
KMeans<> k;
k.Cluster(data, clusters, assignments, centroids);
```

Note that the centroids matrix has columns equal to the number of clusters and rows equal to the dimensionality of the dataset. Each column represents the centroid of the according cluster – `centroids.col(0)` represents the centroid of the first cluster.

11.4.3 Limiting the maximum number of iterations

The first argument to the constructor allows specification of the maximum number of iterations. This is useful because often, the k-means algorithm does not converge, and is terminated after a number of iterations. Setting this parameter to 0 indicates that the algorithm will run until convergence – note that in some cases, convergence may never happen. The default maximum number of iterations is 1000.

```
// The first argument is the maximum number of iterations.  Here we set it to
// 500 iterations.
KMeans<> k(500);
```

Then you can run `Cluster()` as normal.

11.4.4 Setting the overclustering factor

For a description of what overclustering is, see **the command-line interface tutorial about overclustering** (p. 41).

The overclustering factor, which by default is 1.0 (this indicates that no overclustering is happening), is specified in the second argument to the constructor.

```
// We will keep the default maximum iterations of 1000, but set the
// overclustering factor to 2.5.
KMeans<> k(1000, 2.5);
```

Then you can run `Cluster()` as normal.

11.4.5 Setting initial cluster assignments

If you have an initial guess for the cluster assignments for each point, you can fill the assignments vector with the guess and then pass an extra boolean (initialAssignmentGuess) as true to the `Cluster()` method. Below are examples for either overload of `Cluster()`.

```
#include <mlpack/methods/kmeans/kmeans.hpp>

using namespace mlpack::kmeans;

// The dataset we are clustering on.
extern arma::mat dataset;
// The number of clusters we are obtaining.
extern size_t clusters;

// A vector pre-filled with initial assignment guesses.
extern arma::Col<size_t> assignments;

KMeans<> k;

// The boolean set to true indicates that our assignments vector is filled with
// initial guesses.
k.Cluster(dataset, clusters, assignments, true);
```

```
#include <mlpack/methods/kmeans/kmeans.hpp>

using namespace mlpack::kmeans;

// The dataset we are clustering on.
extern arma::mat dataset;
// The number of clusters we are obtaining.
extern size_t clusters;

// A vector pre-filled with initial assignment guesses.
extern arma::Col<size_t> assignments;

// This will hold the centroids of the finished clusters.
arma::mat centroids;
```

```
KMeans<> k;

// The boolean set to true indicates that our assignments vector is filled with
// initial guesses.
k.Cluster(dataset, clusters, assignments, centroids, true);
```

Note

If you have a heuristic or algorithm which makes initial guesses, a more elegant solution is to create a new class fulfilling the InitialPartitionPolicy template policy. See **the section about changing the initial partitioning strategy** (p. 46) for more details.

Note

If you set the InitialPartitionPolicy parameter to something other than the default but give an initial cluster assignment guess, the InitialPartitionPolicy will not be used to initialize the algorithm. See **the section about changing the initial partitioning strategy** (p. 46) for more details.

11.4.6 Setting initial cluster centroids

An equally important option to being able to make initial cluster assignment guesses is to make initial cluster centroid guesses without having to assign each point in the dataset to an initial cluster. This is similar to the previous section, but now you must pass two extra booleans – the first (initialAssignmentGuess) as false, indicating that there are not initial cluster assignment guesses, and the second (initialCentroidGuess) as true, indicating that the centroids matrix is filled with initial centroid guesses.

This, of course, only works with the overload of `Cluster()` that takes a matrix to put the resulting centroids in. Below is an example.

```
#include <mlpack/methods/kmeans/kmeans.hpp>

using namespace mlpack::kmeans;

// The dataset we are clustering on.
extern arma::mat dataset;
// The number of clusters we are obtaining.
extern size_t clusters;

// A matrix pre-filled with guesses for the initial cluster centroids.
extern arma::mat centroids;

// This will be filled with the final cluster assignments for each point.
arma::Col<size_t> assignments;

KMeans<> k;

// Remember, the first boolean indicates that we are not giving initial
// assignment guesses, and the second boolean indicates that we are giving
// initial centroid guesses.
k.Cluster(dataset, clusters, assignments, centroids, false, true);
```

Note

If you have a heuristic or algorithm which makes initial guesses, a more elegant solution is to create a new class fulfilling the InitialPartitionPolicy template policy. See **the section about changing the initial partitioning strategy** (p. 46) for more details.

Note

If you set the InitialPartitionPolicy parameter to something other than the default but give an initial cluster centroid guess, the InitialPartitionPolicy will not be used to initialize the algorithm. See **the section about changing the initial partitioning strategy** (p. 46) for more details.

11.4.7 Running sparse k-means

The `Cluster()` function can work on both sparse and dense matrices, so all of the above examples can be used with sparse matrices instead. Below is a simple example. Note that the centroids are returned as a sparse matrix also.

```
// The sparse dataset.
extern arma::sp_mat sparseDataset;
// The number of clusters.
extern size_t clusters;

// The assignments will be stored in this vector.
arma::Col<size_t> assignments;
// The centroids of each cluster will be stored in this sparse matrix.
arma::sp_mat sparseCentroids;

// No template parameter modification is necessary.
KMeans<> k;
k.Cluster(sparseDataset, clusters, assignments, sparseCentroids);
```

11.5 Template parameters for the 'KMeans' class

The `KMeans<>` class also takes three template parameters, which can be modified to change the behavior of the k-means algorithm. There are three template parameters:

- `MetricType`: controls the distance metric used for clustering (by default, the squared Euclidean distance is used)
- `InitialPartitionPolicy`: the method by which initial clusters are set; by default, **RandomPartition** (p. 267) is used
- `EmptyClusterPolicy`: the action taken when an empty cluster is encountered; by default, **MaxVarianceNewCluster** (p. 266) is used

The class is defined like below:

```
template<
  typename DistanceMetric = mlpack::metric::SquaredEuclideanDistance,
  typename InitialPartitionPolicy = RandomPartition,
  typename EmptyClusterPolicy = MaxVarianceNewCluster
>
class KMeans;
```

In the following sections, each policy is described further, with examples of how to modify them.

11.5.1 Changing the distance metric used for k-means

Most machine learning algorithms in **mlpack** support modifying the distance metric, and `KMeans<>` is no exception. Similar to **NeighborSearch** (p. 317) (see **the section in the NeighborSearch tutorial** (p. 59)), any class in **mlpack::metric** (p. 101) can be given as an argument. The **mlpack::metric::LMetric** (p. 288) class is a good example implementation.

A class fulfilling the MetricType policy must provide the following two functions:

```
// Empty constructor is required.
MetricType();

// Computer the distance between two points.
template<typename VecType>
double Evaluate(const VecType& a, const VecType& b);
```

Most of the standard metrics that could be used are stateless and therefore the `Evaluate()` method is implemented statically. However, there are metrics, such as the Mahalanobis distance (**mlpack::metric::MahalanobisDistance** (p. 290)), that store state. To this end, an instantiated MetricType object is stored within the `KMeans` class. The example below shows how to pass an instantiated MahalanobisDistance in the constructor.

```
// The initialized Mahalanobis distance.
extern mlpack::metric::MahalanobisDistance distance;

// We keep the default arguments for the maximum number of iterations and
// overclustering factor, but pass our instantiated metric.
KMeans<mlpack::metric::MahalanobisDistance> k(1000, 1.0, distance);
```

Note

While the MetricType policy only requires two methods, one of which is an empty constructor, more can always be added. **mlpack::metric::MahalanobisDistance** (p. 290) also has constructors with parameters, because it is a stateful metric.

11.5.2 Changing the initial partitioning strategy used for k-means

There have been many initial cluster strategies for k-means proposed in the literature. Fortunately, the `KMeans<>` class makes it very easy to implement one of these methods and plug it in without needing to modify the existing algorithm code at all.

By default, the `KMeans<>` class uses **mlpack::kmeans::RandomPartition** (p. 267), which randomly partitions points into clusters. However, writing a new policy is simple; it needs to only implement the following functions:

```
// Empty constructor is required.
InitialPartitionPolicy();

// This function is called to initialize the clusters.
template<typename MatType>
void Cluster(MatType& data,
             const size_t clusters,
             arma::Col<size_t> assignments);
```

The templatization of the `Cluster()` function allows both dense and sparse matrices to be passed in. If the desired policy does not work with sparse (or dense) matrices, then the method can be written specifically for one type of matrix – however, be warned that if you try to use `KMeans` with that policy and the wrong type of matrix, you will get many ugly compilation errors!

```
// The Cluster() function specialized for dense matrices.
void Cluster(arma::mat& data,
             const size_t clusters,
             arma::Col<size_t> assignments);
```

One alternate to the default RandomPartition policy is the RefinedStart policy, which is an implementation of the Bradley and Fayyad approach for finding initial points detailed in "Refined initial points for k-means clustering" and other places in this document. Also see the documentation for **mlpack::kmeans::RefinedStart** (p. 269) for more information.

The `Cluster()` method must return valid initial assignments for every point in the dataset.

As with the MetricType template parameter, an initialized InitialPartitionPolicy can be passed to the constructor of `KMeans` as a fourth argument.

11.5.3 Changing the action taken when an empty cluster is encountered

Sometimes, during clustering, a situation will arise where a cluster has no points in it. The `KMeans` class allows easy customization of the action to be taken when this occurs. By default, the point furthest from the centroid of the cluster with maximum variance is taken as the centroid of the empty cluster; this is implemented in the **mlpack::kmeans::MaxVarianceNewCluster** (p. 266) class. Another alternate choice is the **mlpack::kmeans::AllowEmptyClusters** (p. 259) class, which simply allows empty clusters to persist.

A custom policy can be written and it must implement the following methods:

```
// Empty constructor is required.
EmptyClusterPolicy();

// This function is called when an empty cluster is encountered.  emptyCluster
// indicates the cluster which is empty, and then the clusterCounts and
// assignments are meant to be modified by the function.  The function should
// return the number of modified points.
template<typename MatType>
size_t EmptyCluster(const MatType& data,
                    const size_t emptyCluster,
                    const MatType& centroids,
                    arma::Col<size_t>& clusterCounts,
                    arma::Col<size_t>& assignments);
```

The `EmptyCluster()` function is called for each cluster that is empty at each iteration of the algorithm. As with InitialPartitionPolicy, the `EmptyCluster()` function does not need to be generalized to support both dense and sparse matrices – but usage with the wrong type of matrix will cause compilation errors.

Like the other template parameters to `KMeans`, EmptyClusterPolicy implementations that have state can be passed to the constructor of `KMeans` as a fifth argument. See the kmeans::KMeans documentation for further details.

11.6 Further documentation

For further documentation on the KMeans class, consult the **complete API documentation** (p. 260).

Chapter 12

Linear/ridge regression tutorial (linear_regression)

12.1 Introduction

Linear regression and ridge regression are simple machine learning techniques that aim to estimate the parameters of a linear model. Assuming we have n **predictor** points $\mathbf{x_i}, 0 \leq i < n$ of dimensionality d and n responses $y_i, 0 \leq i < n$, we are trying to estimate the best fit for $\beta_i, 0 \leq i \leq d$ in the linear model

$$y_i = \beta_0 + \sum_{j=1}^{d} \beta_j x_{ij}$$

for each predictor $\mathbf{x_i}$ and response y_i. If we take each predictor $\mathbf{x_i}$ as a row in the matrix $\mathbf{X}$ and each response y_i as an entry of the vector $\mathbf{y}$, we can represent the model in vector form:

$$\mathbf{y} = \mathbf{X}\beta + \beta_0$$

The result of this method is the vector β, including the offset term (or intercept term) β_0.

mlpack provides:

- a **simple command-line executable** (p. 50) to perform linear regression or ridge regression
- a **simple C++ interface** (p. 53) to perform linear regression or ridge regression

12.2 Table of Contents

A list of all the sections this tutorial contains.

12.3 Command-Line 'linear_regression'

The simplest way to perform linear regression or ridge regression in **mlpack** is to use the linear_regression executable. This program will perform linear regression and place the resultant coefficients into one file.

The output file holds a vector of coefficients in increasing order of dimension; that is, the offset term (β_0), the coefficient for dimension 1 (β_1, then dimension 2 (β_2) and so forth, as well as the intercept. This executable can also predict the y values of a second dataset based on the computed coefficients.

Below are several examples of simple usage (and the resultant output). The '-v' option is used so that verbose output is given. Further documentation on each individual option can be found by typing

```
$ linear_regression --help
```

12.3.1 One file, generating the function coefficients

```
$ linear_regression --input_file dataset.csv -v
[INFO ] Loading 'dataset.csv' as CSV data.
[INFO ] Saving CSV data to 'parameters.csv'.
[INFO ]
[INFO ] Execution parameters:
[INFO ]   help: false
[INFO ]   info: ""
[INFO ]   input_file: dataset.csv
[INFO ]   input_responses: ""
[INFO ]   lambda: 0
[INFO ]   output_file: parameters.csv
[INFO ]   output_predictions: predictions.csv
[INFO ]   test_file: ""
[INFO ]   verbose: true
[INFO ]
[INFO ] Program timers:
[INFO ]   load_regressors: 0.006461s
[INFO ]   regression: 0.000347s
[INFO ]   total_time: 0.026589s
```

Convenient program timers are given for different parts of the calculation at the bottom of the output, as well as the parameters the simulation was run with. Now, if we look at the output file, which, unless specified, is parameters.csv:

```
$ cat dataset.csv
0,0
1,1
2,2
3,3
4,4

$ cat parameters.csv
-0.0000000000e+00,1.0000000000e+00
```

As you can see, the function for this input is $f(y) = 0 + 1x_1$. Keep in mind that in this example, the regressors for the dataset are the second column. That is, the dataset is one dimensional, and the last column has the y values, or responses, for each row. You can specify these responses in a separate file if you want, using the `--input_responses`, or `-r`, option.

12.3.2 Compute model and predict at the same time

```
$ linear_regression --input_file dataset.csv --test_file predict.csv -v
[INFO ] Loading 'dataset.csv' as CSV data.
[INFO ] Saving CSV data to 'parameters.csv'.
[INFO ] Loading 'predict.csv' as CSV data.
[INFO ] Saving CSV data to 'predictions.csv'.
[INFO ]
[INFO ] Execution parameters:
[INFO ]   help: false
[INFO ]   info: ""
[INFO ]   input_file: dataset.csv
[INFO ]   input_responses: ""
[INFO ]   lambda: 0
[INFO ]   model_file: ""
[INFO ]   output_file: parameters.csv
[INFO ]   output_predictions: predictions.csv
[INFO ]   test_file: predict.csv
[INFO ]   verbose: true
[INFO ]
[INFO ] Program timers:
[INFO ]   load_regressors: 0.000360s
[INFO ]   load_test_points: 0.000090s
[INFO ]   prediction: 0.000006s
[INFO ]   regression: 0.000335s
[INFO ]   total_time: 0.001522s

$ cat dataset.csv
0,0
1,1
2,2
3,3
4,4

$ cat parameters.csv
-0.0000000000e+00,1.0000000000e+00

$ cat predict.csv
2
3
4

$ cat predictions.csv
2.0000000000e+00
3.0000000000e+00
4.0000000000e+00
```

We used the same dataset, so we got the same parameters. The key thing to note about the predict.csv dataset is that it has the same dimensionality as the dataset used to create the model, one. Generally, if the model generating dataset has d dimensions, so must the dataset we want to predict for.

12.3.3 Prediction using a precomputed model

```
$ linear_regression --model_file parameters.csv --test_file predict.csv -v
[INFO ] Loading 'parameters.csv' as CSV data.
[INFO ] Loading 'predict.csv' as CSV data.
[INFO ] Saving CSV data to 'predictions.csv'.
[INFO ]
[INFO ] Execution parameters:
[INFO ]   help: false
[INFO ]   info: ""
[INFO ]   input_file: ""
[INFO ]   input_responses: ""
[INFO ]   lambda: 0
[INFO ]   model_file: parameters.csv
[INFO ]   output_file: parameters.csv
```

```
[INFO ]   output_predictions: predictions.csv
[INFO ]   test_file: predict.csv
[INFO ]   verbose: true
[INFO ]
[INFO ] Program timers:
[INFO ]   load_model: 0.009519s
[INFO ]   load_test_points: 0.000067s
[INFO ]   prediction: 0.000007s
[INFO ]   total_time: 0.010081s

$ cat parameters.csv
-0.0000000000e+00,1.0000000000e+00

$ cat predict.csv
2
3
4

$ cat predictions.csv
2.0000000000e+00
3.0000000000e+00
4.0000000000e+00
```

12.3.4 Using ridge regression

Sometimes, the input matrix of predictors has a covariance matrix that is not invertible, or the system is overdetermined. In this case, ridge regression is useful: it adds a normalization term to the covariance matrix to make it invertible. Ridge regression is a standard technique and documentation for the mathematics behind it can be found anywhere on the Internet. In short, the covariance matrix

$$\mathbf{X}'\mathbf{X}$$

is replaced with

$$\mathbf{X}'\mathbf{X} + \lambda \mathbf{I}$$

where $\mathbf{I}$ is the identity matrix. So, a λ parameter greater than zero should be specified to perform ridge regression, using the `-lambda` (or `-l`) option. An example is given below.

```
$ linear_regression --input_file dataset.csv -v --lambda 0.5
[INFO ] Loading 'dataset.csv' as CSV data.  Size is 3 x 1000.
[INFO ] Saving CSV data to 'parameters.csv'.
[INFO ]
[INFO ] Execution parameters:
[INFO ]   help: false
[INFO ]   info: ""
[INFO ]   input_file: test_data_3_1000.csv
[INFO ]   input_responses: ""
[INFO ]   lambda: 0.5
[INFO ]   model_file: ""
[INFO ]   output_file: parameters.csv
[INFO ]   output_predictions: predictions.csv
[INFO ]   test_file: ""
[INFO ]   verbose: true
[INFO ]
[INFO ] Program timers:
[INFO ]   load_regressors: 0.005236s
[INFO ]   loading_data: 0.005208s
[INFO ]   regression: 0.013206s
[INFO ]   saving_data: 0.000276s
[INFO ]   total_time: 0.020019s
```

Further documentation on options should be found by using the `-help` option.

12.4 The 'LinearRegression' class

The 'LinearRegression' class is a simple implementation of linear regression.

Using the LinearRegression class is very simple. It has two available constructors; one for generating a model from a matrix of predictors and a vector of responses, and one for loading an already computed model from a given file.

The class provides one method that performs computation:

```
void Predict(const arma::mat& points, arma::vec& predictions);
```

Once you have generated or loaded a model, you can call this method and pass it a matrix of data points to predict values for using the model. The second parameter, predictions, will be modified to contain the predicted values corresponding to each row of the points matrix.

12.4.1 Generating a model

```
#include <mlpack/methods/linear_regression/linear_regression.hpp>

using namespace mlpack::regression;

arma::mat data; // The dataset itself.
arma::vec responses; // The responses, one row for each row in data.

// Regress.
LinearRegression lr(data,responses);

// Get the parameters, or coefficients.
arma::vec parameters = lr.Parameters();
```

12.4.2 Setting a model

Assuming you already have a model and do not need to create one, this is how you would set the parameters for a LinearRegression instance.

```
arma::vec parameters; // Your model.

LinearRegression lr(); // Create a new LinearRegression instance or reuse one.
lr.Parameters() = parameters; // Set the model.
```

12.4.3 Load a model from a file

If you have a generated model in a file somewhere you would like to load and use, you can simply pass it to the LinearRegression initializer like so.

```
std::string filename; // The path and name of your file.

LinearRegression lr(filename); // Will load the model internally.
```

12.4.4 Prediction

Once you have generated or loaded a model using one of the aforementioned methods, you can predict values for a dataset.

```
LinearRegression lr();
// Load or generate your model.

// The dataset we want to predict on; each row is a data point.
```

```
arma::mat points;
// This will store the predictions; one row for each point.
arma::vec predictions;

lr.Predict(points, predictions); // Predict.

// Now, the vector 'predictions' will contain the predicted values.
```

12.4.5 Setting lambda for ridge regression

As discussed in **Using ridge regression** (p. 52), ridge regression is useful when the covariance of the predictors is not invertible. The standard constructor can be used to set a value of lambda:

```
#include <mlpack/methods/linear_regression/linear_regression.hpp>

using namespace mlpack::regression;

arma::mat data; // The dataset itself.
arma::vec responses; // The responses, one row for each row in data.

// Regress, with a lambda of 0.5.
LinearRegression lr(data, responses, 0.5);

// Get the parameters, or coefficients.
arma::vec parameters = lr.Parameters();
```

In addition, the `Lambda()` function can be used to get or modify the lambda value:

```
LinearRegression lr;
lr.Lambda() = 0.5;
Log::Info << "Lambda is " << lr.Lambda() << "." << std::endl;
```

12.5 Further documentation

For further documentation on the LinearRegression class, consult the **complete API documentation** (p. 435).

Chapter 13

NeighborSearch tutorial (k-nearest-neighbors)

13.1 Introduction

Nearest-neighbors search is a common machine learning task. In this setting, we have a **query** and a **reference** dataset. For each point in the **query** dataset, we wish to know the k points in the **reference** dataset which are closest to the given query point.

Alternately, if the query and reference datasets are the same, the problem can be stated more simply: for each point in the dataset, we wish to know the k nearest points to that point.

mlpack provides:

- a **simple command-line executable** (p. 56) to run nearest-neighbors search (and furthest-neighbors search)
- a **simple C++ interface** (p. 58) to perform nearest-neighbors search (and furthest-neighbors search)
- a **generic, extensible, and powerful C++ class (NeighborSearch)** (p. 59) for complex usage

13.2 Table of Contents

A list of all the sections this tutorial contains.

13.3 Command-Line 'allknn'

The simplest way to perform nearest-neighbors search in **mlpack** is to use the allknn executable. This program will perform nearest-neighbors search and place the resultant neighbors into one file and the resultant distances into another. The output files are organized such that the first row corresponds to the nearest neighbors of the first query point, with the first column corresponding to the nearest neighbor, and so forth.

Below are several examples of simple usage (and the resultant output). The '-v' option is used so that output is given. Further documentation on each individual option can be found by typing

```
$ allknn --help
```

13.3.1 One dataset, 5 nearest neighbors

```
$ allknn -r dataset.csv -n neighbors_out.csv -d distances_out.csv -k 5 -v
[INFO ] Loading 'dataset.csv' as CSV data.
[INFO ] Loaded reference data from 'dataset.csv'.
[INFO ] Building reference tree...
[INFO ] Trees built.
[INFO ] Computing 5 nearest neighbors...
[INFO ] Neighbors computed.
[INFO ] Re-mapping indices...
[INFO ] Saving CSV data to 'distances_out.csv'.
[INFO ] Saving CSV data to 'neighbors_out.csv'.
[INFO ]
[INFO ] Execution parameters:
[INFO ]   distances_file: distances_out.csv
[INFO ]   help: false
[INFO ]   info: ""
[INFO ]   k: 5
[INFO ]   leaf_size: 20
[INFO ]   naive: false
[INFO ]   neighbors_file: neighbors_out.csv
[INFO ]   query_file: ""
[INFO ]   reference_file: dataset.csv
[INFO ]   single_mode: false
[INFO ]   verbose: true
[INFO ]
[INFO ] Program timers:
[INFO ]   computing_neighbors: 0.152495s
[INFO ]   total_time: 0.201274s
[INFO ]   tree_building: 0.005050s
```

Convenient program timers are given for different parts of the calculation at the bottom of the output, as well as the parameters the simulation was run with. Now, if we look at the output files:

```
$ head neighbors_out.csv
14,5,13,16,27
90,79,80,15,10
39,84,10,123,1
81,43,109,12,37
15,1,79,90,10
0,14,16,13,27
90,79,11,1,15
41,45,12,37,49
11,81,13,6,15
41,7,45,49,47

$ head distances_out.csv
```

```
7.09614421e-04,2.05940173e-03,4.05346068e-03,4.66175278e-03,1.09757665e-02
8.92190948e-04,1.69442242e-03,2.82750475e-03,4.06590850e-03,7.54169243e-03
5.91539406e-03,6.83482612e-03,8.02877800e-03,9.04907425e-03,1.61458442e-02
7.15652913e-03,9.18228524e-03,1.00540941e-02,1.07541171e-02,1.28892864e-02
5.37535983e-03,9.05721409e-03,9.89017184e-03,1.01457735e-02,1.14021593e-02
2.05940173e-03,5.14437192e-03,9.97483954e-03,1.02463627e-02,1.44355783e-02
4.27355419e-03,6.36750547e-03,6.72478577e-03,8.77323532e-03,1.04530549e-02
1.99935847e-03,3.88240331e-03,4.19118273e-03,9.30693568e-03,1.21237481e-02
2.15454276e-03,8.18895210e-03,1.18360450e-02,1.25135454e-02,1.27783327e-02
8.43087996e-03,1.22946325e-02,1.60472209e-02,1.88661413e-02,1.89727686e-02
```

So, the nearest neighbor to point 0 is point 14, with a distance of 7.096144e-4. The second nearest neighbor to point 0 is point 5, with a distance of 2.059402e-3. The third nearest neighbor to point 5 is point 16, with a distance of 9.9748395e-3.

13.3.2 Query and reference dataset, 10 nearest neighbors

```
$ allknn -q query_dataset.csv -r reference_dataset.csv -n neighbors_out.csv \
> -d distances_out.csv -k 10 -v
[INFO ] Loading 'reference_dataset.csv' as CSV data.
[INFO ] Loaded reference data from 'reference_dataset.csv'.
[INFO ] Building reference tree...
[INFO ] Loading 'query_dataset.csv' as CSV data.
[INFO ] Query data loaded from 'query_dataset.csv'.
[INFO ] Building query tree...
[INFO ] Tree built.
[INFO ] Computing 10 nearest neighbors...
[INFO ] Neighbors computed.
[INFO ] Re-mapping indices...
[INFO ] Saving CSV data to 'distances_out.csv'.
[INFO ] Saving CSV data to 'neighbors_out.csv'.
[INFO ]
[INFO ] Execution parameters:
[INFO ]   distances_file: distances_out.csv
[INFO ]   help: false
[INFO ]   info: ""
[INFO ]   k: 10
[INFO ]   leaf_size: 20
[INFO ]   naive: false
[INFO ]   neighbors_file: neighbors_out.csv
[INFO ]   query_file: query_dataset.csv
[INFO ]   reference_file: reference_dataset.csv
[INFO ]   single_mode: false
[INFO ]   verbose: true
[INFO ]
[INFO ] Program timers:
[INFO ]   computing_neighbors: 0.000081s
[INFO ]   total_time: 0.062828s
[INFO ]   tree_building: 0.004949s
```

13.3.3 One dataset, 3 nearest neighbors, leaf size of 15 points

```
$ allknn -r dataset.csv -n neighbors_out.csv -d distances_out.csv -k 3 -l 15 -v
[INFO ] Loading 'dataset.csv' as CSV data.
[INFO ] Loaded reference data from 'dataset.csv'.
[INFO ] Building reference tree...
[INFO ] Trees built.
[INFO ] Computing 3 nearest neighbors...
[INFO ] Neighbors computed.
[INFO ] Re-mapping indices...
[INFO ] Saving CSV data to 'distances_out.csv'.
[INFO ] Saving CSV data to 'neighbors_out.csv'.
[INFO ]
[INFO ] Execution parameters:
[INFO ]   distances_file: distances_out.csv
[INFO ]   help: false
[INFO ]   info: ""
[INFO ]   k: 3
[INFO ]   leaf_size: 15
[INFO ]   naive: false
[INFO ]   neighbors_file: neighbors_out.csv
[INFO ]   query_file: ""
[INFO ]   reference_file: dataset.csv
```

```
[INFO ]   single_mode: false
[INFO ]   verbose: true
[INFO ]
[INFO ] Program timers:
[INFO ]   computing_neighbors: 0.105119s
[INFO ]   total_time: 0.145321s
[INFO ]   tree_building: 0.005690s
```

Further documentation on options should be found by using the –help option.

13.4 The 'AllkNN' class

The 'AllkNN' class is, specifically, a typedef of the more extensible NeighborSearch class, querying for nearest neighbors using the squared Euclidean distance.

```
typedef NeighborSearch<NearestNeighborSort, metric::SquaredEuclideanDistance>
    AllkNN;
```

Using the AllkNN class is particularly simple; first, the object must be constructed and given a dataset. Then, the method is run, and two matrices are returned: one which holds the indices of the nearest neighbors, and one which holds the distances of the nearest neighbors. These are of the same structure as the output –neighbors_file and –distances_file for the CLI interface (see above). A handful of examples of simple usage of the AllkNN class are given below.

13.4.1 5 nearest neighbors on a single dataset

```
#include <mlpack/methods/neighbor_search/neighbor_search.hpp>

using namespace mlpack::neighbor;

// Our dataset matrix, which is column-major.
extern arma::mat data;

AllkNN a(data);

// The matrices we will store output in.
arma::Mat<size_t> resultingNeighbors;
arma::mat resultingDistances;

a.Search(5, resultingNeighbors, resultingDistances);
```

The output of the search is stored in resultingNeighbors and resultingDistances.

13.4.2 10 nearest neighbors on a query and reference dataset

```
#include <mlpack/methods/neighbor_search/neighbor_search.hpp>

using namespace mlpack::neighbor;

// Our dataset matrices, which are column-major.
extern arma::mat queryData, referenceData;

AllkNN a(referenceData, queryData);

// The matrices we will store output in.
arma::Mat<size_t> resultingNeighbors;
arma::mat resultingDistances;

a.Search(10, resultingNeighbors, resultingDistances);
```

13.4.3 Naive (exhaustive) search for 6 nearest neighbors on one dataset

This example uses the O(n^2) naive search (not the tree-based search).

```
#include <mlpack/methods/neighbor_search/neighbor_search.hpp>

using namespace mlpack::neighbor;

// Our dataset matrix, which is column-major.
extern arma::mat dataset;

AllkNN a(dataset, true);

// The matrices we will store output in.
arma::Mat<size_t> resultingNeighbors;
arma::mat resultingDistances;

a.Search(6, resultingNeighbors, resultingDistances);
```

Needless to say, naive search can be very slow...

13.5 The extensible 'NeighborSearch' class

The NeighborSearch class is very extensible, having the following template arguments:

```
template<
  typename SortPolicy = NearestNeighborSort,
  typename MetricType = mlpack::metric::SquaredEuclideanDistance,
  typename TreeType = mlpack::tree::BinarySpaceTree<bound::HRectBound<2>,
                                                    QueryStat<SortPolicy> >
>
class NeighborSearch;
```

By choosing different components for each of these template classes, a very arbitrary neighbor searching object can be constructed.

13.5.1 SortPolicy policy class

The SortPolicy template parameter allows specification of how the NeighborSearch object will decide which points are to be searched for. The **mlpack::neighbor::NearestNeighborSort** (p. 314) class is a well-documented example. A custom SortPolicy class must implement the same methods which NearestNeighborSort does:

```
static size_t SortDistance(const arma::vec& list, double newDistance);

static bool IsBetter(const double value, const double ref);

template<typename TreeType>
static double BestNodeToNodeDistance(const TreeType* queryNode,
                                     const TreeType* referenceNode);

template<typename TreeType>
static double BestPointToNodeDistance(const arma::vec& queryPoint,
                                      const TreeType* referenceNode);

static const double WorstDistance();

static const double BestDistance();
```

The **mlpack::neighbor::FurthestNeighborSort** (p. 304) class is another implementation, which is used to create the 'AllkFN' typedef class, which finds the furthest neighbors, as opposed to the nearest neighbors.

13.5.2 MetricType policy class

The MetricType policy class allows the neighbor search to take place in any arbitrary metric space. The **mlpack::metric::LMetric** (p. 288) class is a good example implementation. A MetricType class must provide the following functions:

```
// Empty constructor is required.
MetricType();

// Compute the distance between two points.
template<typename VecType>
double Evaluate(const VecType& a, const VecType& b);
```

Internally, the NeighborSearch class keeps an instantiated MetricType class (which can be given in the constructor). This is useful for a metric like the Mahalanobis distance (**mlpack::metric::MahalanobisDistance** (p. 290)), which must store state (the covariance matrix). Therefore, you can write a non-static MetricType class and use it seamlessly with NeighborSearch.

13.5.3 TreeType policy class

The NeighborSearch class also allows a custom tree to be used. The standard MLPACK tree, **mlpack::tree::BinarySpaceTree** (p. 459), is also highly extensible in its own right, and its documentation should be consulted for more information. Currently, the NeighborSearch tree requires a tree which only has left and right children, and no points in nodes (only in leaves), but this support is planned to be extended.

A simple usage of the TreeType policy could be to use a different type of bound with the tree. For instance, you could use a ball bound instead of a rectangular bound:

```
// Construct a NeighborSearch object with ball bounds.
NeighborSearch<
  NearestNeighborSort,
  metric::SquaredEuclideanDistance,
  tree::BinarySpaceTree<bound::BallBound<2>,
                        QueryStat<SortPolicy> >
> neighborSearch(dataset);
```

It is important to note that the NeighborSearch class requires use of the QueryStat tree statistic to function properly. Therefore, if you write a custom tree, be sure it can accept the QueryStat type. See the **mlpack::tree::BinarySpaceTree** (p. 459) documentation for more information on tree statistics.

13.6 Further documentation

For further documentation on the NeighborSearch class, consult the **complete API documentation** (p. 317).

Chapter 14

RangeSearch tutorial (range_search)

14.1 Introduction

Range search is a simple machine learning task which aims to find all the neighbors of a point that fall into a certain range of distances. In this setting, we have a **query** and a **reference** dataset. Given a certain range, for each point in the **query** dataset, we wish to know all points in the **reference** dataset which have distances within that given range to the given query point.

Alternately, if the query and reference datasets are the same, the problem can be stated more simply: for each point in the dataset, we wish to know all points which have distance in the given range to that point.

mlpack provides:

- a **simple command-line executable** (p. 62) to run range search
- a **simple C++ interface** (p. 64) to perform range search
- a **generic, extensible, and powerful C++ class (RangeSearch)** (p. 65) for complex usage

14.2 Table of Contents

A list of all the sections this tutorial contains.

14.3 The 'range_search' command-line executable

mlpack provides an exectuable, range_search, which can be used to perform range searches quickly and simply from the command-line. This program will perform the range search and place the resulting neighbor index list into one file and their corresponding distances into another file. These files are organized such that the first row corresponds to the neighbors (or distances) of the first query point, and the second row corresponds to the neighbors (or distances) of the second query point, and so forth. The neighbors of a specific point are not arranged in any specific order.

Because a range search may return different numbers of points (including zero), the output file is technically not a valid CSV and may not be loadable by other programs. Therefore, if you need the results in a certain format, it may be better to use the **C++ interface** (p. 64) to manually export the data in the preferred format.

Below are several examples of simple usage (and the resultant output). The '-v' option is used so that output is given. Further documentation on each individual option can be found by typing

```
$ range_search --help
```

14.3.1 One dataset, points with distance <= 0.01

```
$ range_search -r dataset.csv -n neighbors_out.csv -d distances_out.csv -M 0.01 -v
[INFO ] Loading 'dataset.csv' as CSV data.
[INFO ] Loaded reference data from 'dataset.csv'.
[INFO ] Building reference tree...
[INFO ] Trees built.
[INFO ] Computing neighbors within range [0, 0.01].
[INFO ] Number of pruned nodes during computation: 0.
[INFO ] Neighbors computed.
[INFO ] Re-mapping indices...
[INFO ]
[INFO ] Execution parameters:
[INFO ]   distances_file: distances_out.csv
[INFO ]   help: false
[INFO ]   info: ""
[INFO ]   leaf_size: 20
[INFO ]   max: 2.5
[INFO ]   min: 0
[INFO ]   naive: false
[INFO ]   neighbors_file: neighbors_out.csv
[INFO ]   query_file: ""
[INFO ]   reference_file: dataset.csv
[INFO ]   single_mode: false
[INFO ]   verbose: true
[INFO ]
[INFO ] Program timers:
[INFO ]   range_search/computing_neighbors: 1.564744s
[INFO ]   total_time: 3.841249s
[INFO ]   tree_building: 0.005112s
```

Convenient program timers are given for different parts of the calculation at the bottom of the output, as well as the parameters the simulation was run with. Now, if we look at the output files:

```
$ head neighbors_out.csv
344, 862
703

397, 277, 319, 443
840, 827
```

```
876, 732
569, 222, 563
437, 361, 97, 928
961, 419, 547, 695
113, 843, 634, 982, 689

$ head distances_out.csv
0.0058751, 0.00358331
0.00567406

0.000432393, 0.00577239, 0.00221909, 0.00841252
0.00501577, 0.00810424
0.00898339, 0.0032354
0.00945658, 0.00893871, 0.006213
0.00979697, 0.00490745, 0.00833828, 0.00902167
0.00957553, 0.00657434, 0.0028044, 0.00303588
0.00199936, 0.00843088, 0.00968861, 0.00159429, 0.00539645
```

We can see that points 344 and 862 are within distance 0.01 of point 0. We can also see that point 2 has no points within a distance of 0.01 – that line is empty.

14.3.2 Query and reference dataset, range [1.0, 1.5]

```
$ range_search -q query_dataset.csv -r reference_dataset.csv -n \
> neighbors_out.csv -d distances_out.csv -m 1.0 -M 1.5 -v
[INFO ] Loading 'dataset.csv' as CSV data.
[INFO ] Loaded reference data from 'dataset.csv'.
[INFO ] Building reference tree...
[INFO ] Loading 'dataset.csv' as CSV data.
[INFO ] Loaded query data from 'dataset.csv'.
[INFO ] Building query tree...
[INFO ] Tree built.
[INFO ] Computing neighbors within range [1, 1.5].
[INFO ] Number of pruned nodes during computation: 1110.
[INFO ] Neighbors computed.
[INFO ] Re-mapping indices...
[INFO ]
[INFO ] Execution parameters:
[INFO ]   distances_file: distances_out.csv
[INFO ]   help: false
[INFO ]   info: ""
[INFO ]   leaf_size: 20
[INFO ]   max: 1.5
[INFO ]   min: 1
[INFO ]   naive: false
[INFO ]   neighbors_file: neighbors_out.csv
[INFO ]   query_file: dataset.csv
[INFO ]   reference_file: dataset.csv
[INFO ]   single_mode: false
[INFO ]   verbose: true
[INFO ]
[INFO ] Program timers:
[INFO ]   range_search/computing_neighbors: 0.466848s
[INFO ]   total_time: 0.725183s
[INFO ]   tree_building: 0.004769s
```

14.3.3 One dataset, range [4.1 4.2], leaf size of 15 points

The **mlpack** implementation of range search is a dual-tree method, meaning that the leaf size of the tree can be changed. Depending on the characteristics of the dataset, a larger or smaller leaf size can provide faster computation. The leaf size is modifiable through the command-line interface, as shown below.

```
$ range_search -r dataset.csv -n neighbors_out.csv -d distances_out.csv -m 4.1 \
> -M 4.2 -l 15 -v
[INFO ] Loading 'dataset.csv' as CSV data.
[INFO ] Loaded reference data from 'dataset.csv'.
[INFO ] Building reference tree...
[INFO ] Trees built.
[INFO ] Computing neighbors within range [4.1, 4.2].
[INFO ] Number of pruned nodes during computation: 1.
[INFO ] Neighbors computed.
```

```
[INFO ] Re-mapping indices...
[INFO ]
[INFO ] Execution parameters:
[INFO ]   distances_file: distances_out.csv
[INFO ]   help: false
[INFO ]   info: ""
[INFO ]   leaf_size: 20
[INFO ]   max: 4.2
[INFO ]   min: 4.1
[INFO ]   naive: false
[INFO ]   neighbors_file: neighbors_out.csv
[INFO ]   query_file: ""
[INFO ]   reference_file: dataset.csv
[INFO ]   single_mode: false
[INFO ]   verbose: true
[INFO ]
[INFO ] Program timers:
[INFO ]   range_search/computing_neighbors: 0.003857s
[INFO ]   total_time: 0.056154s
[INFO ]   tree_building: 0.004831s
```

Further documentation on options should be found by using the –help option.

14.4 The 'RangeSearch' class

The 'RangeSearch' class is an extensible template class which allows a high level of flexibility. However, all of the template arguments have default parameters, allowing a user to simply use 'RangeSearch<>' for simple usage without worrying about the exact necessary template parameters.

The class bears many similarities to the **NeighborSearch** (p. 55) class; usage generally consists of calling the constructor with one or two datasets, and then calling the 'Search()' method to perform the actual range search.

The 'Search()' method stores the results in two vector-of-vector objects. This is necessary because each query point may have a different number of neighbors in the specified distance range. The structure of those two objects is very similar to the output files –neighbors_file and –distances_file for the CLI interface (see above). A handful of examples of simple usage of the RangeSearch class are given below.

Using the AllkNN class is particularly simple; first, the object must be constructed and given a dataset. Then, the method is run, and two matrices are returned: one which holds the indices of the nearest neighbors, and one which holds the distances of the nearest neighbors. These are of the same structure as the output –neighbors_file and –reference_file for the CLI interface (see above). A handful of examples of simple usage of the AllkNN class are given below.

14.4.1 Distance less than 2.0 on a single dataset

```
#include <mlpack/methods/range_search/range_search.hpp>

using namespace mlpack::range;

// Our dataset matrix, which is column-major.
extern arma::mat data;

RangeSearch<> a(data);

// The vector-of-vector objects we will store output in.
std::vector<std::vector<size_t> > resultingNeighbors;
std::vector<std::vector<double> > resultingDistances;

// The range we will use.
math::Range r(0.0, 2.0); // [0.0, 2.0].

a.Search(r, resultingNeighbors, resultingDistances);
```

The output of the search is stored in resultingNeighbors and resultingDistances.

14.4.2 Range [3.0, 4.0] on a query and reference dataset

```
#include <mlpack/methods/range_search/range_search.hpp>

using namespace mlpack::range;

// Our dataset matrices, which are column-major.
extern arma::mat queryData, referenceData;

RangeSearch<> a(referenceData, queryData);

// The vector-of-vector objects we will store output in.
std::vector<std::vector<size_t> > resultingNeighbors;
std::vector<std::vector<double> > resultingDistances;

// The range we will use.
math::Range r(3.0, 4.0); // [3.0, 4.0].

a.Search(r, resultingNeighbors, resultingDistances);
```

14.4.3 Naive (exhaustive) search for distance greater than 5.0 on one dataset

This example uses the O(n^2) naive search (not the tree-based search).

```
#include <mlpack/methods/range_search/range_search.hpp>

using namespace mlpack::range;

// Our dataset matrix, which is column-major.
extern arma::mat dataset;

// The 'true' option indicates that we will use naive calculation.
RangeSearch<> a(dataset, true);

// The vector-of-vector objects we will store output in.
std::vector<std::vector<size_t> > resultingNeighbors;
std::vector<std::vector<double> > resultingDistances;

// The range we will use.  The upper bound is DBL_MAX.
math::Range r(5.0, DBL_MAX); // [5.0, inf).

a.Search(r, resultingNeighbors, resultingDistances);
```

Needless to say, naive search can be very slow...

14.5 The extensible 'RangeSearch' class

Similar to the **NeighborSearch class** (p. 55), the RangeSearch class is very extensible, having the following template arguments:

```
template<
  typename MetricType = mlpack::metric::SquaredEuclideanDistance,
  typename TreeType = mlpack::tree::BinarySpaceTree<bound::HRectBound<2>,
                                                    tree::EmptyStatistic>
>
class RangeSearch;
```

By choosing different components for each of these template classes, a very arbitrary range searching object can be constructed.

14.5.1 MetricType policy class

The MetricType policy class allows the range search to take place in any arbitrary metric space. The **mlpack::metric::LMetric** (p. 288) class is a good example implementation. A MetricType class must provide the following functions:

```
// Empty constructor is required.
MetricType();

// Compute the distance between two points.
template<typename VecType>
double Evaluate(const VecType& a, const VecType& b);
```

Internally, the RangeSearch class keeps an instantiated MetricType class (which can be given in the constructor). This is useful for a metric like the Mahalanobis distance (**mlpack::metric::MahalanobisDistance** (p. 290)), which must store state (the covariance matrix). Therefore, you can write a non-static MetricType class and use it seamlessly with RangeSearch.

14.5.2 TreeType policy class

The RangeSearch class also allows a custom tree to be used. The standard **mlpack** tree, **mlpack::tree::BinarySpaceTree** (p. 459), is also highly extensible in its own right, and its documentation should be consulted for more information. Currently, the RangeSearch tree requires a tree which only has left and right children, and no points in nodes (only in leaves), but this support is planned to be extended.

A simple usage of the TreeType policy could be to use a different type of bound with the tree. For instance, you could use a ball bound instead of a rectangular bound:

```
// Construct a NeighborSearch object with ball bounds.
RangeSearch<
  metric::SquaredEuclideanDistance,
  tree::BinarySpaceTree<bound::BallBound<2>,
                        EmptyStatistic>
> rangeSearch(dataset);
```

Unlike the **NeighborSearch class** (p. 55), the RangeSearch class does not make use of tree statistics; therefore, the EmptyStatistic class should be used for the StatisticType parameter of the BinarySpaceTree (but this is not technically necessary – RangeSearch simply makes no use of the tree statistic).

14.6 Further documentation

For further documentation on the RangeSearch class, consult the **complete API documentation** (p. 415).

Chapter 15

Tutorials

15.1 Introductory Tutorials

These tutorials introduce the basic concepts of working with MLPACK, aimed at developers who want to use and contribute to MLPACK but are not sure where to start.

- **Building MLPACK From Source** (p. 5)
- **Matrices in MLPACK** (p. 13)
- **MLPACK Input and Output** (p. 9)
- **MLPACK Timers** (p. 17)
- **Simple Sample MLPACK Programs** (p. 15)

15.2 Method-specific Tutorials

These tutorials introduce the various methods MLPACK offers, aimed at users who simply want to use the methods MLPACK offers. These tutorials start with simple examples and progress to complex, extensible uses.

- **NeighborSearch tutorial (k-nearest-neighbors)** (p. 55)
- **Linear/ridge regression tutorial (linear_regression)** (p. 49)
- **RangeSearch tutorial (range_search)** (p. 61)
- **Density Estimation Tree (DET) tutorial** (p. 21)
- **K-Means tutorial (kmeans)** (p. 39)
- **Fast max-kernel search tutorial (fastmks)** (p. 31)
- **EMST Tutorial** (p. 27)

Chapter 16

Bug List

Class mlpack::CLI (p. **132**)

The **COUNTER** variable is used in most cases to guarantee a unique global identifier for options declared using the PARAM_∗() macros. However, not all compilers have this support–most notably, gcc $<$ 4.3. In that case, the **LINE** macro is used as an attempt to get a unique global identifier, but collisions are still possible, and they produce bizarre error messages. See `http://mlpack.org/trac/ticket/74` for more information.

Member PARAM_DOUBLE (p. **625**) **(ID, DESC, ALIAS, DEF)**

The **COUNTER** variable is used in most cases to guarantee a unique global identifier for options declared using the PARAM_∗() macros. However, not all compilers have this support–most notably, gcc $<$ 4.3. In that case, the **LINE** macro is used as an attempt to get a unique global identifier, but collisions are still possible, and they produce bizarre error messages. See `http://mlpack.org/trac/ticket/74` for more information.

Member PARAM_DOUBLE_REQ (p. **625**) **(ID, DESC, ALIAS)**

The **COUNTER** variable is used in most cases to guarantee a unique global identifier for options declared using the PARAM_∗() macros. However, not all compilers have this support–most notably, gcc $<$ 4.3. In that case, the **LINE** macro is used as an attempt to get a unique global identifier, but collisions are still possible, and they produce bizarre error messages. See `http://mlpack.org/trac/ticket/74` for more information.

Member PARAM_FLAG (p. **626**) **(ID, DESC, ALIAS)**

The **COUNTER** variable is used in most cases to guarantee a unique global identifier for options declared using the PARAM_∗() macros. However, not all compilers have this support–most notably, gcc $<$ 4.3. In that case, the **LINE** macro is used as an attempt to get a unique global identifier, but collisions are still possible, and they produce bizarre error messages. See `http://mlpack.org/trac/ticket/74` for more information.

Member PARAM_FLOAT (p. **626**) **(ID, DESC, ALIAS, DEF)**

The **COUNTER** variable is used in most cases to guarantee a unique global identifier for options declared using the PARAM_∗() macros. However, not all compilers have this support–most notably, gcc $<$ 4.3. In that case, the **LINE** macro is used as an attempt to get a unique global identifier, but collisions are still possible, and they produce bizarre error messages. See `http://mlpack.org/trac/ticket/74` for more information.

Member PARAM_FLOAT_REQ (p. **627**) **(ID, DESC, ALIAS)**

The **COUNTER** variable is used in most cases to guarantee a unique global identifier for options declared using the PARAM_∗() macros. However, not all compilers have this support–most notably, gcc $<$ 4.3. In that case, the **LINE** macro is used as an attempt to get a unique global identifier, but collisions are still possible, and they produce bizarre error messages. See `http://mlpack.org/trac/ticket/74` for more information.

Member PARAM_INT (p. **627**) **(ID, DESC, ALIAS, DEF)**

The **COUNTER** variable is used in most cases to guarantee a unique global identifier for options declared using the PARAM_∗() macros. However, not all compilers have this support–most notably, gcc $<$ 4.3. In that case, the **LINE** macro is used as an attempt to get a unique global identifier, but collisions are still possible, and they produce bizarre error messages. See `http://mlpack.org/trac/ticket/74` for more information.

Member PARAM_INT_REQ (p. 628) (ID, DESC, ALIAS)

The **COUNTER** variable is used in most cases to guarantee a unique global identifier for options declared using the PARAM_∗() macros. However, not all compilers have this support–most notably, gcc $<$ 4.3. In that case, the **LINE** macro is used as an attempt to get a unique global identifier, but collisions are still possible, and they produce bizarre error messages. See `http://mlpack.org/trac/ticket/74` for more information.

Member PARAM_STRING (p. 628) (ID, DESC, ALIAS, DEF)

The **COUNTER** variable is used in most cases to guarantee a unique global identifier for options declared using the PARAM_∗() macros. However, not all compilers have this support–most notably, gcc $<$ 4.3. In that case, the **LINE** macro is used as an attempt to get a unique global identifier, but collisions are still possible, and they produce bizarre error messages. See `http://mlpack.org/trac/ticket/74` for more information.

Member PARAM_STRING_REQ (p. 629) (ID, DESC, ALIAS)

The **COUNTER** variable is used in most cases to guarantee a unique global identifier for options declared using the PARAM_∗() macros. However, not all compilers have this support–most notably, gcc $<$ 4.3. In that case, the **LINE** macro is used as an attempt to get a unique global identifier, but collisions are still possible, and they produce bizarre error messages. See `http://mlpack.org/trac/ticket/74` for more information.

Member PARAM_VECTOR (p. 629) (T, ID, DESC, ALIAS)

The **COUNTER** variable is used in most cases to guarantee a unique global identifier for options declared using the PARAM_∗() macros. However, not all compilers have this support–most notably, gcc $<$ 4.3. In that case, the **LINE** macro is used as an attempt to get a unique global identifier, but collisions are still possible, and they produce bizarre error messages. See `http://mlpack.org/trac/ticket/74` for more information.

Member PARAM_VECTOR_REQ (p. 630) (T, ID, DESC, ALIAS)

The **COUNTER** variable is used in most cases to guarantee a unique global identifier for options declared using the PARAM_∗() macros. However, not all compilers have this support–most notably, gcc $<$ 4.3. In that case, the **LINE** macro is used as an attempt to get a unique global identifier, but collisions are still possible, and they produce bizarre error messages. See `http://mlpack.org/trac/ticket/74` for more information.

Chapter 17

Namespace Index

17.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 18

Class Index

18.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 19

File Index

19.1 File List

Here is a list of all files with brief descriptions:

Chapter 20

Namespace Documentation

20.1 mlpack Namespace Reference

Linear algebra utility functions, generally performed on matrices or vectors.

Namespaces

- **bound**
- **cf**

  Collaborative filtering.
- **data**

  Functions to load and save matrices.
- **det**

  Density Estimation Trees.
- **distribution**

  Probability distributions.
- **emst**

  Euclidean Minimum Spanning Trees.
- **fastmks**

  Fast max-kernel search.
- **gmm**

  Gaussian Mixture Models.
- **hmm**

  Hidden Markov Models.
- **kernel**

  Kernel functions.
- **kmeans**

  K-Means clustering.
- **kpca**
- **lcc**
- **math**

  Miscellaneous math routines.
- **metric**
- **naive_bayes**

The Naive Bayes Classifier.

- **nca**

  Neighborhood Components Analysis.
- **neighbor**

  Neighbor-search routines.
- **nmf**
- **optimization**
- **pca**
- **radical**
- **range**

  Range-search routines.
- **regression**

  Regression methods.
- **sparse_coding**
- **tree**

  Trees and tree-building procedures.
- **util**

Classes

- class **CLI**

  Parses the command line for parameters and holds user-specified parameters.
- class **Log**

  Provides a convenient way to give formatted output.
- struct **ParamData**

  *Aids in the extensibility of **CLI** (*p. *132) by focusing potential changes into one structure.*
- class **Timer**

  The timer class provides a way for MLPACK methods to be timed.
- class **Timers**

20.1.1 Detailed Description

Linear algebra utility functions, generally performed on matrices or vectors.

20.2 mlpack::bound Namespace Reference

Classes

- class **BallBound**

  Ball bound that works in the regular Euclidean metric space.
- class **HRectBound**

  Hyper-rectangle bound for an L-metric.
- class **PeriodicHRectBound**

  Hyper-rectangle bound for an L-metric.

20.3 mlpack::cf Namespace Reference

Collaborative filtering.

Classes

- class **CF**

  *This class implements Collaborative Filtering (**CF** (p. 126)).*

20.3.1 Detailed Description

Collaborative filtering.

20.4 mlpack::data Namespace Reference

Functions to load and save matrices.

Functions

- template<typename eT >
  bool **Load** (const std::string &filename, arma::Mat< eT > &matrix, bool fatal=false, bool transpose=true)

  Loads a matrix from file, guessing the filetype from the extension.
- template<typename eT >
  void **NormalizeLabels** (const arma::Col< eT > &labelsIn, arma::Col< size_t > &labels, arma::Col< eT > &mapping)

  Given a set of labels of a particular datatype, convert them to unsigned labels in the range [0, n) where n is the number of different labels.
- template<typename eT >
  void **RevertLabels** (const arma::Col< size_t > &labels, const arma::Col< eT > &mapping, arma::Col< eT > &labelsOut)

  Given a set of labels that have been mapped to the range [0, n), map them back to the original labels given by the 'mapping' vector.
- template<typename eT >
  bool **Save** (const std::string &filename, const arma::Mat< eT > &matrix, bool fatal=false, bool transpose=true)

  Saves a matrix to file, guessing the filetype from the extension.

20.4.1 Detailed Description

Functions to load and save matrices.

20.4.2 Function Documentation

20.4.2.1 template<typename eT > bool mlpack::data::Load (const std::string & *filename,* arma::Mat< eT > & *matrix,* bool *fatal* = `false`, bool *transpose* = `true`)

Loads a matrix from file, guessing the filetype from the extension.

This will transpose the matrix at load time. If the filetype cannot be determined, an error will be given.

The supported types of files are the same as found in Armadillo:

- CSV (csv_ascii), denoted by .csv, or optionally .txt
- ASCII (raw_ascii), denoted by .txt
- Armadillo ASCII (arma_ascii), also denoted by .txt
- PGM (pgm_binary), denoted by .pgm
- PPM (ppm_binary), denoted by .ppm
- Raw binary (raw_binary), denoted by .bin
- Armadillo binary (arma_binary), denoted by .bin
- HDF5, denoted by .hdf, .hdf5, .h5, or .he5

If the file extension is not one of those types, an error will be given. This is preferable to Armadillo's default behavior of loading an unknown filetype as raw_binary, which can have very confusing effects.

If the parameter 'fatal' is set to true, the program will exit with an error if the matrix does not load successfully. The parameter 'transpose' controls whether or not the matrix is transposed after loading. In most cases, because data is generally stored in a row-major format and MLPACK requires column-major matrices, this should be left at its default value of 'true'.

Parameters

filename	Name of file to load.
matrix	Matrix to load contents of file into.
fatal	If an error should be reported as fatal (default false).
transpose	If true, transpose the matrix after loading.

Returns

Boolean value indicating success or failure of load.

20.4.2.2 template<typename eT > void mlpack::data::NormalizeLabels (const arma::Col< eT > & *labelsIn,* arma::Col< size_t > & *labels,* arma::Col< eT > & *mapping*)

Given a set of labels of a particular datatype, convert them to unsigned labels in the range [0, n) where n is the number of different labels.

Also, a reverse mapping from the new label to the old value is stored in the 'mapping' vector.

Parameters

labelsIn	Input labels of arbitrary datatype.
labels	Vector that unsigned labels will be stored in.
mapping	Reverse mapping to convert new labels back to old labels.

20.4.2.3 template<typename eT > void mlpack::data::RevertLabels (const arma::Col< size_t > & *labels,* const arma::Col< eT > & *mapping,* arma::Col< eT > & *labelsOut*)

Given a set of labels that have been mapped to the range [0, n), map them back to the original labels given by the 'mapping' vector.

Parameters

labels	Set of normalized labels to convert.
mapping	Mapping to use to convert labels.
labelsOut	Vector to store new labels in.

20.4.2.4 template<typename eT > bool mlpack::data::Save (const std::string & *filename,* const arma::Mat< eT > & *matrix,* bool *fatal* = `false`, bool *transpose* = `true`)

Saves a matrix to file, guessing the filetype from the extension.

This will transpose the matrix at save time. If the filetype cannot be determined, an error will be given.

The supported types of files are the same as found in Armadillo:

- CSV (csv_ascii), denoted by .csv, or optionally .txt
- ASCII (raw_ascii), denoted by .txt
- Armadillo ASCII (arma_ascii), also denoted by .txt
- PGM (pgm_binary), denoted by .pgm
- PPM (ppm_binary), denoted by .ppm
- Raw binary (raw_binary), denoted by .bin
- Armadillo binary (arma_binary), denoted by .bin
- HDF5 (hdf5_binary), denoted by .hdf5, .hdf, .h5, or .he5

If the file extension is not one of those types, an error will be given. If the 'fatal' parameter is set to true, an error will cause the program to exit. If the 'transpose' parameter is set to true, the matrix will be transposed before saving. Generally, because MLPACK stores matrices in a column-major format and most datasets are stored on disk as row-major, this parameter should be left at its default value of 'true'.

Parameters

filename	Name of file to save to.
matrix	Matrix to save into file.
fatal	If an error should be reported as fatal (default false).
transpose	If true, transpose the matrix before saving.

Returns

Boolean value indicating success or failure of save.

20.5 mlpack::det Namespace Reference

Density Estimation Trees.

Classes

- class **DTree**

  A density estimation tree is similar to both a decision tree and a space partitioning tree (like a kd-tree).

Functions

- void **PrintLeafMembership** (**DTree** *dtree, const arma::mat &data, const arma::Mat< size_t > &labels, const size_t numClasses, const std::string leafClassMembershipFile="")

  Print the membership of leaves of a density estimation tree given the labels and number of classes.
- void **PrintVariableImportance** (const **DTree** *dtree, const std::string viFile="")

  Print the variable importance of each dimension of a density estimation tree.
- **DTree** * **Trainer** (arma::mat &dataset, const size_t folds, const bool useVolumeReg=false, const size_t maxLeafSize=10, const size_t minLeafSize=5, const std::string unprunedTreeOutput="")

  Train the optimal decision tree using cross-validation with the given number of folds.

20.5.1 Detailed Description

Density Estimation Trees.

20.5.2 Function Documentation

20.5.2.1 void mlpack::det::PrintLeafMembership (DTree * *dtree,* const arma::mat & *data,* const arma::Mat< size_t > & *labels,* const size_t *numClasses,* const std::string *leafClassMembershipFile* = " ")

Print the membership of leaves of a density estimation tree given the labels and number of classes.

Optionally, pass the name of a file to print this information to (otherwise stdout is used).

Parameters

dtree	Tree to print membership of.
data	Dataset tree is built upon.
labels	Class labels of dataset.
numClasses	Number of classes in dataset.
leafClass-MembershipFile	Name of file to print to (optional).

20.5.2.2 void mlpack::det::PrintVariableImportance (const DTree * *dtree,* const std::string *viFile* = " ")

Print the variable importance of each dimension of a density estimation tree.

Optionally, pass the name of a file to print this information to (otherwise stdout is used).

Parameters

dtree	Density tree to use.
viFile	Name of file to print to (optional).

20.5.2.3 DTree* mlpack::det::Trainer (arma::mat & *dataset,* const size_t *folds,* const bool *useVolumeReg* = `false`, const size_t *maxLeafSize* = `10`, const size_t *minLeafSize* = `5`, const std::string *unprunedTreeOutput* = " ")

Train the optimal decision tree using cross-validation with the given number of folds.

Optionally, give a filename to print the unpruned tree to. This initializes a tree on the heap, so you are responsible for deleting it.

Parameters

dataset	Dataset for the tree to use.
folds	Number of folds to use for cross-validation.
useVolumeReg	If true, use volume regularization.
maxLeafSize	Maximum number of points allowed in a leaf.
minLeafSize	Minimum number of points allowed in a leaf.
unprunedTree-Output	Filename to print unpruned tree to (optional).

20.6 mlpack::distribution Namespace Reference

Probability distributions.

Classes

- class **DiscreteDistribution**

  A discrete distribution where the only observations are discrete observations.
- class **GaussianDistribution**

  A single multivariate Gaussian distribution.

20.6.1 Detailed Description

Probability distributions.

20.7 mlpack::emst Namespace Reference

Euclidean Minimum Spanning Trees.

Classes

- class **DTBRules**
- class **DTBStat**

  A statistic for use with MLPACK trees, which stores the upper bound on distance to nearest neighbors and the component which this node belongs to.
- class **DualTreeBoruvka**

  Performs the MST calculation using the Dual-Tree Boruvka algorithm, using any type of tree.
- class **EdgePair**

  An edge pair is simply two indices and a distance.
- class **UnionFind**

  A Union-Find data structure.

20.7.1 Detailed Description

Euclidean Minimum Spanning Trees.

20.8 mlpack::fastmks Namespace Reference

Fast max-kernel search.

Classes

- class **FastMKS**

  An implementation of fast exact max-kernel search.
- class **FastMKSRules**

  *The base case and pruning rules for **FastMKS** (p. 185) (fast max-kernel search).*
- class **FastMKSStat**

  *The statistic used in trees with **FastMKS** (p. 185).*

20.8.1 Detailed Description

Fast max-kernel search.

20.9 mlpack::gmm Namespace Reference

Gaussian Mixture Models.

Classes

- class **DiagonalConstraint**

  Force a covariance matrix to be diagonal.
- class **EigenvalueRatioConstraint**

  Given a vector of eigenvalue ratios, ensure that the covariance matrix always has those eigenvalue ratios.
- class **EMFit**

  *This class contains methods which can fit a **GMM** (p. 208) to observations using the EM algorithm.*
- class **GMM**

  *A Gaussian Mixture Model (**GMM** (p. 208)).*
- class **NoConstraint**

  This class enforces no constraint on the covariance matrix.
- class **PositiveDefiniteConstraint**

  Given a covariance matrix, force the matrix to be positive definite.

Functions

- double **phi** (const double x, const double mean, const double var)

  Calculates the univariate Gaussian probability density function.
- double **phi** (const arma::vec &x, const arma::vec &mean, const arma::mat &cov)

  Calculates the multivariate Gaussian probability density function.
- double **phi** (const arma::vec &x, const arma::vec &mean, const arma::mat &cov, const std::vector< arma::mat > &d_cov, arma::vec &g_mean, arma::vec &g_cov)

Calculates the multivariate Gaussian probability density function and also the gradients with respect to the mean and the variance.

- void **phi** (const arma::mat &x, const arma::vec &mean, const arma::mat &cov, arma::vec &probabilities)

  Calculates the multivariate Gaussian probability density function for each data point (column) in the given matrix, with respect to the given mean and variance.

20.9.1 Detailed Description

Gaussian Mixture Models.

20.9.2 Function Documentation

20.9.2.1 double mlpack::gmm::phi (const double *x*, const double *mean*, const double *var*) [inline]

Calculates the univariate Gaussian probability density function.

Example use:

```
double x, mean, var;
....
double f = phi(x, mean, var);
```

Parameters

x	Observation.
mean	Mean of univariate Gaussian.
var	Variance of univariate Gaussian.

Returns

Probability of x being observed from the given univariate Gaussian.

Definition at line 46 of file phi.hpp.

References M_PI.

Referenced by mlpack::distribution::GaussianDistribution::Probability().

20.9.2.2 double mlpack::gmm::phi (const arma::vec & *x*, const arma::vec & *mean*, const arma::mat & *cov*) [inline]

Calculates the multivariate Gaussian probability density function.

Example use:

```
extern arma::vec x, mean;
extern arma::mat cov;
....
double f = phi(x, mean, cov);
```

Parameters

x	Observation.
mean	Mean of multivariate Gaussian.
cov	Covariance of multivariate Gaussian.

Returns

Probability of x being observed from the given multivariate Gaussian.

Definition at line 68 of file phi.hpp.

References M_PI.

20.9.2.3 double mlpack::gmm::phi (const arma::vec & *x*, const arma::vec & *mean*, const arma::mat & *cov*, const std::vector< arma::mat > & *d_cov*, arma::vec & *g_mean*, arma::vec & *g_cov*) [inline]

Calculates the multivariate Gaussian probability density function and also the gradients with respect to the mean and the variance.

Example use:

```
extern arma::vec x, mean, g_mean, g_cov;
std::vector<arma::mat> d_cov; // the dSigma
....
double f = phi(x, mean, cov, d_cov, &g_mean, &g_cov);
```

Definition at line 94 of file phi.hpp.

References M_PI.

20.9.2.4 void mlpack::gmm::phi (const arma::mat & *x*, const arma::vec & *mean*, const arma::mat & *cov*, arma::vec & *probabilities*) [inline]

Calculates the multivariate Gaussian probability density function for each data point (column) in the given matrix, with respect to the given mean and variance.

Parameters

x	List of observations.
mean	Mean of multivariate Gaussian.
cov	Covariance of multivariate Gaussian.
probabilities	Output probabilities for each input observation.

Definition at line 138 of file phi.hpp.

References M_PI.

20.10 mlpack::hmm Namespace Reference

Hidden Markov Models.

Classes

- class **HMM**

  A class that represents a Hidden Markov Model with an arbitrary type of emission distribution.

Functions

- template<typename Distribution >
  void **LoadHMM** (**HMM**< Distribution > &hmm, **util::SaveRestoreUtility** &sr)

  *Load an **HMM** (p. 221) from file.*

- template<typename Distribution >
  void **SaveHMM** (const **HMM**< Distribution > &hmm, **util::SaveRestoreUtility** &sr)

  *Save an **HMM** (p. 221) to file.*

20.10.1 Detailed Description

Hidden Markov Models.

20.10.2 Function Documentation

20.10.2.1 template<typename Distribution > void mlpack::hmm::LoadHMM (HMM< Distribution > & *hmm,* util::SaveRestoreUtility & *sr*)

Load an **HMM** (p. 221) from file.

This only works for GMMs, DiscreteDistributions, and GaussianDistributions.

Template Parameters

Distribution	Distribution type of **HMM** (p. 221).

Parameters

sr	SaveRestoreUtility to use.

20.10.2.2 template<typename Distribution > void mlpack::hmm::SaveHMM (const HMM< Distribution > & *hmm,* util::SaveRestoreUtility & *sr*)

Save an **HMM** (p. 221) to file.

This only works for GMMs, DiscreteDistributions, and GaussianDistributions.

Template Parameters

Distribution	Distribution type of **HMM** (p. 221).

Parameters

sr	SaveRestoreUtility to use.

20.11 mlpack::kernel Namespace Reference

Kernel functions.

Classes

- class **CosineDistance**

  The cosine distance (or cosine similarity).
- class **EpanechnikovKernel**

  The Epanechnikov kernel, defined as.
- class **ExampleKernel**

  An example kernel function.
- class **GaussianKernel**

  The standard Gaussian kernel.
- class **HyperbolicTangentKernel**

  Hyperbolic tangent kernel.
- class **KernelTraits**

  This is a template class that can provide information about various kernels.
- class **KernelTraits**< **CosineDistance** >

  Kernel traits for the cosine distance.
- class **KernelTraits**< **EpanechnikovKernel** >

  Kernel traits for the Epanechnikov kernel.
- class **KernelTraits**< **GaussianKernel** >

  Kernel traits for the Gaussian kernel.
- class **KernelTraits**< **LaplacianKernel** >

  Kernel traits of the Laplacian kernel.
- class **KernelTraits**< **SphericalKernel** >

  Kernel traits for the spherical kernel.
- class **KernelTraits**< **TriangularKernel** >

  Kernel traits for the triangular kernel.
- class **LaplacianKernel**

  The standard Laplacian kernel.
- class **LinearKernel**

  The simple linear kernel (dot product).
- class **PolynomialKernel**

  The simple polynomial kernel.
- class **PSpectrumStringKernel**

  The p-spectrum string kernel.
- class **SphericalKernel**
- class **TriangularKernel**

  The trivially simple triangular kernel, defined by.

20.11.1 Detailed Description

Kernel functions. This namespace contains kernel functions, which evaluate some kernel function $K(x,y)$ for some arbitrary vectors x and y of the same dimension. The single restriction on the function $K(x,y)$ is that it must satisfy Mercer's condition:

$$\int \int K(x,y)g(x)g(y)dxdy \geq 0$$

for all square integrable functions $g(x)$.

The kernels in this namespace all implement the same methods as the **ExampleKernel** (p. 232) class. Any additional custom kernels should implement all the methods that class implements; in addition, any method using a kernel should rely on any arbitrary kernel function class having a default constructor and a function

```
double Evaluate(arma::vec&, arma::vec&);
```

20.12 mlpack::kmeans Namespace Reference

K-Means clustering.

Classes

- class **AllowEmptyClusters**

  Policy which allows K-Means to create empty clusters without any error being reported.
- class **KMeans**

  This class implements K-Means clustering.
- class **MaxVarianceNewCluster**

  When an empty cluster is detected, this class takes the point furthest from the centroid of the cluster with maximum variance as a new cluster.
- class **RandomPartition**

  A very simple partitioner which partitions the data randomly into the number of desired clusters.
- class **RefinedStart**

  A refined approach for choosing initial points for k-means clustering.

20.12.1 Detailed Description

K-Means clustering.

20.13 mlpack::kpca Namespace Reference

Classes

- class **KernelPCA**

  This class performs kernel principal components analysis (Kernel PCA), for a given kernel.

20.14 mlpack::lcc Namespace Reference

Classes

- class **LocalCoordinateCoding**

  An implementation of Local Coordinate Coding (LCC) that codes data which approximately lives on a manifold using a variation of l1-norm regularized sparse coding; in LCC, the penalty on the absolute value of each point's coefficient for each atom is weighted by the squared distance of that point to that atom.

20.15 mlpack::math Namespace Reference

Miscellaneous math routines.

Classes

- class **Range**

  Simple real-valued range.

Functions

- void **Center** (const arma::mat &x, arma::mat &xCentered)

  Creates a centered matrix, where centering is done by subtracting the sum over the columns (a column vector) from each column of the matrix.
- double **ClampNonNegative** (const double d)

  Forces a number to be non-negative, turning negative numbers into zero.
- double **ClampNonPositive** (const double d)

  Forces a number to be non-positive, turning positive numbers into zero.
- double **ClampRange** (double value, const double rangeMin, const double rangeMax)

  Clamp a number between a particular range.
- void **Orthogonalize** (const arma::mat &x, arma::mat &W)

  Orthogonalize x and return the result in W, using eigendecomposition.
- void **Orthogonalize** (arma::mat &x)

  Orthogonalize x in-place.
- int **RandInt** (const int hiExclusive)

  Generates a uniform random integer.
- int **RandInt** (const int lo, const int hiExclusive)

  Generates a uniform random integer.
- double **RandNormal** ()

  Generates a normally distributed random number with mean 0 and variance 1.
- double **RandNormal** (const double mean, const double variance)

  Generates a normally distributed random number with specified mean and variance.
- double **Random** ()

  Generates a uniform random number between 0 and 1.
- double **Random** (const double lo, const double hi)

  Generates a uniform random number in the specified range.
- void **RandomSeed** (const size_t seed)

  *Set the random seed used by the random functions (**Random()** (p. 99) and **RandInt()** (p. 98)).*
- void **RandVector** (arma::vec &v)

  Overwrites a dimension-N vector to a random vector on the unit sphere in R^N.
- void **RemoveRows** (const arma::mat &input, const std::vector< size_t > &rowsToRemove, arma::mat &output)

  Remove a certain set of rows in a matrix while copying to a second matrix.
- void **VectorPower** (arma::vec &vec, double power)

  Auxiliary function to raise vector elements to a specific power.
- void **WhitenUsingEig** (const arma::mat &x, arma::mat &xWhitened, arma::mat &whiteningMatrix)

  Whitens a matrix using the eigendecomposition of the covariance matrix.
- void **WhitenUsingSVD** (const arma::mat &x, arma::mat &xWhitened, arma::mat &whiteningMatrix)

  Whitens a matrix using the singular value decomposition of the covariance matrix.

Variables

- boost::mt19937 **randGen**
- boost::normal_distribution **randNormalDist**
- boost::uniform_01
  < boost::mt19937, double > **randUniformDist**

20.15.1 Detailed Description

Miscellaneous math routines.

20.15.2 Function Documentation

20.15.2.1 void mlpack::math::Center (const arma::mat & *x*, arma::mat & *xCentered*)

Creates a centered matrix, where centering is done by subtracting the sum over the columns (a column vector) from each column of the matrix.

Parameters

x	Input matrix
xCentered	Matrix to write centered output into

20.15.2.2 double mlpack::math::ClampNonNegative (const double *d*) [inline]

Forces a number to be non-negative, turning negative numbers into zero.

Avoids branching costs (this is a measurable improvement).

Parameters

d	Double to clamp.

Returns

0 if d < 0, d otherwise.

Definition at line 38 of file clamp.hpp.

Referenced by ClampRange().

20.15.2.3 double mlpack::math::ClampNonPositive (const double *d*) [inline]

Forces a number to be non-positive, turning positive numbers into zero.

Avoids branching costs (this is a measurable improvement).

Parameters

d	Double to clamp.

0	if d $>$ 0, d otherwise.

Definition at line 50 of file clamp.hpp.

Referenced by ClampRange().

20.15.2.4 double mlpack::math::ClampRange (double *value,* const double *rangeMin,* const double *rangeMax*) [inline]

Clamp a number between a particular range.

Parameters

value	The number to clamp.
rangeMin	The first of the range.
rangeMax	The last of the range.

Returns

max(rangeMin, min(rangeMax, d)).

Definition at line 63 of file clamp.hpp.

References ClampNonNegative(), and ClampNonPositive().

20.15.2.5 void mlpack::math::Orthogonalize (const arma::mat & *x,* arma::mat & *W*)

Orthogonalize x and return the result in W, using eigendecomposition.

We will be using the formula $W = x(x^T x)^{-0.5}$.

20.15.2.6 void mlpack::math::Orthogonalize (arma::mat & *x*)

Orthogonalize x in-place.

This could be sped up by a custom implementation.

20.15.2.7 int mlpack::math::RandInt (const int *hiExclusive*) [inline]

Generates a uniform random integer.

Definition at line 103 of file random.hpp.

References randGen, and randUniformDist.

Referenced by mlpack::sparse_coding::DataDependentRandomInitializer::Initialize(), and mlpack::nmf::RandomAcolInitialization$<$ p $>$::Initialize().

20.15.2.8 int mlpack::math::RandInt (const int *lo,* const int *hiExclusive*) [inline]

Generates a uniform random integer.

Definition at line 117 of file random.hpp.

References randGen, and randUniformDist.

20.15.2.9 double mlpack::math::RandNormal ( ) [inline]

Generates a normally distributed random number with mean 0 and variance 1.

Definition at line 134 of file random.hpp.

References randGen, and randNormalDist.

20.15.2.10 double mlpack::math::RandNormal (const double *mean*, const double *variance*) [inline]

Generates a normally distributed random number with specified mean and variance.

Parameters

mean	Mean of distribution.
variance	Variance of distribution.

Definition at line 146 of file random.hpp.

References randGen, and randNormalDist.

20.15.2.11 double mlpack::math::Random ( ) [inline]

Generates a uniform random number between 0 and 1.

Definition at line 75 of file random.hpp.

References randGen, and randUniformDist.

20.15.2.12 double mlpack::math::Random (const double *lo*, const double *hi*) [inline]

Generates a uniform random number in the specified range.

Definition at line 89 of file random.hpp.

References randGen, and randUniformDist.

20.15.2.13 void mlpack::math::RandomSeed (const size_t *seed*) [inline]

Set the random seed used by the random functions (**Random()** (p. 99) and **RandInt()** (p. 98)).

The seed is casted to a 32-bit integer before being given to the random number generator, but a size_t is taken as a parameter for API consistency.

Parameters

seed	Seed for the random number generator.

Definition at line 66 of file random.hpp.

References randGen.

20.15.2.14 void mlpack::math::RandVector (arma::vec & *v*)

Overwrites a dimension-N vector to a random vector on the unit sphere in R^N.

20.15.2.15 void mlpack::math::RemoveRows (const arma::mat & *input,* const std::vector< size_t > & *rowsToRemove,* arma::mat & *output*)

Remove a certain set of rows in a matrix while copying to a second matrix.

Parameters

input	Input matrix to copy.
rowsToRemove	Vector containing indices of rows to be removed.
output	Matrix to copy non-removed rows into.

20.15.2.16 void mlpack::math::VectorPower (arma::vec & *vec,* double *power*)

Auxiliary function to raise vector elements to a specific power.

The sign is ignored in the power operation and then re-added. Useful for eigenvalues.

20.15.2.17 void mlpack::math::WhitenUsingEig (const arma::mat & *x,* arma::mat & *xWhitened,* arma::mat & *whiteningMatrix*)

Whitens a matrix using the eigendecomposition of the covariance matrix.

Whitening means the covariance matrix of the result is the identity matrix.

20.15.2.18 void mlpack::math::WhitenUsingSVD (const arma::mat & *x,* arma::mat & *xWhitened,* arma::mat & *whiteningMatrix*)

Whitens a matrix using the singular value decomposition of the covariance matrix.

Whitening means the covariance matrix of the result is the identity matrix.

20.15.3 Variable Documentation

20.15.3.1 boost::mt19937 mlpack::math::randGen

Referenced by RandInt(), RandNormal(), Random(), and RandomSeed().

20.15.3.2 boost::normal_distribution mlpack::math::randNormalDist

Referenced by RandNormal().

20.15.3.3 boost::uniform_01<boost::mt19937, double> mlpack::math::randUniformDist

Referenced by RandInt(), and Random().

20.16 mlpack::metric Namespace Reference

Classes

- class **IPMetric**
- class **LMetric**

  The L_p metric for arbitrary integer p, with an option to take the root.
- class **MahalanobisDistance**

  The Mahalanobis distance, which is essentially a stretched Euclidean distance.

Typedefs

- typedef **LMetric**< INT_MAX, false > **ChebyshevDistance**
- typedef **LMetric**< 2, true > **EuclideanDistance**
- typedef **LMetric**< 1, false > **ManhattanDistance**
- typedef **LMetric**< 2, false > **SquaredEuclideanDistance**

20.16.1 Typedef Documentation

20.16.1.1 typedef LMetric<INT_MAX, false> mlpack::metric::ChebyshevDistance

Definition at line 109 of file lmetric.hpp.

20.16.1.2 typedef LMetric<2, true> mlpack::metric::EuclideanDistance

Definition at line 104 of file lmetric.hpp.

20.16.1.3 typedef LMetric<1, false> mlpack::metric::ManhattanDistance

Definition at line 94 of file lmetric.hpp.

20.16.1.4 typedef LMetric<2, false> mlpack::metric::SquaredEuclideanDistance

Definition at line 99 of file lmetric.hpp.

20.17 mlpack::naive_bayes Namespace Reference

The Naive Bayes Classifier.

Classes

- class **NaiveBayesClassifier**

  The simple Naive Bayes classifier.

20.17.1 Detailed Description

The Naive Bayes Classifier.

20.18 mlpack::nca Namespace Reference

Neighborhood Components Analysis.

Classes

- class **NCA**

  An implementation of Neighborhood Components Analysis, both a linear dimensionality reduction technique and a distance learning technique.
- class **SoftmaxErrorFunction**

  The "softmax" stochastic neighbor assignment probability function.

20.18.1 Detailed Description

Neighborhood Components Analysis.

20.19 mlpack::neighbor Namespace Reference

Neighbor-search routines.

Classes

- class **FurthestNeighborSort**

  *This class implements the necessary methods for the SortPolicy template parameter of the **NeighborSearch** (p. 317) class.*
- class **LSHSearch**

  *The **LSHSearch** (p. 307) class – This class builds a hash on the reference set and uses this hash to compute the distance-approximate nearest-neighbors of the given queries.*
- class **NearestNeighborSort**

  *This class implements the necessary methods for the SortPolicy template parameter of the **NeighborSearch** (p. 317) class.*
- class **NeighborSearch**

  *The **NeighborSearch** (p. 317) class is a template class for performing distance-based neighbor searches.*
- class **NeighborSearchRules**
- class **NeighborSearchStat**

  Extra data for each node in the tree.
- class **RAQueryStat**

  Extra data for each node in the tree.
- class **RASearch**

  *The **RASearch** (p. 334) class: This class provides a generic manner to perform rank-approximate search via random-sampling.*
- class **RASearchRules**

Typedefs

- typedef **NeighborSearch** $<$ **FurthestNeighborSort**, **metric::EuclideanDistance** $>$ **AllkFN**

  The AllkFN class is the all-k-furthest-neighbors method.
- typedef **NeighborSearch** $<$ **NearestNeighborSort**, **metric::EuclideanDistance** $>$ **AllkNN**

The AllkNN class is the all-k-nearest-neighbors method.

- typedef **RASearch** < **FurthestNeighborSort** > **AllkRAFN**

  The AllkRAFN class is the all-k-rank-approximate-farthest-neighbors method.

- typedef **RASearch AllkRANN**

  The AllkRANN class is the all-k-rank-approximate-nearest-neighbors method.

Functions

- void **Unmap** (const arma::Mat< size_t > &neighbors, const arma::mat &distances, const std::vector< size_t > &referenceMap, const std::vector< size_t > &queryMap, arma::Mat< size_t > &neighborsOut, arma::mat &distancesOut, const bool squareRoot=false)

  Assuming that the datasets have been mapped using the referenceMap and the queryMap (such as during kd-tree construction), unmap the columns of the distances and neighbors matrices into neighborsOut and distancesOut, and also unmap the entries in each row of neighbors.

- void **Unmap** (const arma::Mat< size_t > &neighbors, const arma::mat &distances, const std::vector< size_t > &referenceMap, arma::Mat< size_t > &neighborsOut, arma::mat &distancesOut, const bool squareRoot=false)

  Assuming that the datasets have been mapped using referenceMap (such as during kd-tree construction), unmap the columns of the distances and neighbors matrices into neighborsOut and distancesOut, and also unmap the entries in each row of neighbors.

20.19.1 Detailed Description

Neighbor-search routines. These include all-nearest-neighbors and all-furthest-neighbors searches.

20.19.2 Typedef Documentation

20.19.2.1 typedef NeighborSearch<FurthestNeighborSort, metric::EuclideanDistance> mlpack::neighbor::AllkFN

The AllkFN class is the all-k-furthest-neighbors method.

It returns L2 distances (Euclidean distances) for each of the k furthest neighbors.

Definition at line 48 of file typedef.hpp.

20.19.2.2 typedef NeighborSearch<NearestNeighborSort, metric::EuclideanDistance> mlpack::neighbor::AllkNN

The AllkNN class is the all-k-nearest-neighbors method.

It returns L2 distances (Euclidean distances) for each of the k nearest neighbors.

Definition at line 42 of file typedef.hpp.

20.19.2.3 typedef RASearch<FurthestNeighborSort> mlpack::neighbor::AllkRAFN

The AllkRAFN class is the all-k-rank-approximate-farthest-neighbors method.

It returns squared L2 distances (squared Euclidean distances) for each of the k rank-approximate farthest-neighbors. Squared distances are used because they are slightly faster than non-squared distances (they have one fewer call to sqrt()).

The approximation is controlled with two parameters (see allkrann_main.cpp) which can be specified at search time. So the tree building is done only once while the search can be performed multiple times with different approximation levels.

Definition at line 63 of file ra_typedef.hpp.

20.19.2.4 typedef RASearch mlpack::neighbor::AllkRANN

The AllkRANN class is the all-k-rank-approximate-nearest-neighbors method.

It returns squared L2 distances (squared Euclidean distances) for each of the k rank-approximate nearest-neighbors. Squared distances are used because they are slightly faster than non-squared distances (they have one fewer call to sqrt()).

The approximation is controlled with two parameters (see allkrann_main.cpp) which can be specified at search time. So the tree building is done only once while the search can be performed multiple times with different approximation levels.

Definition at line 49 of file ra_typedef.hpp.

20.19.3 Function Documentation

20.19.3.1 void mlpack::neighbor::Unmap (const arma::Mat< size_t > & *neighbors*, const arma::mat & *distances*, const std::vector< size_t > & *referenceMap*, const std::vector< size_t > & *queryMap*, arma::Mat< size_t > & *neighborsOut*, arma::mat & *distancesOut*, const bool *squareRoot* = `false`)

Assuming that the datasets have been mapped using the referenceMap and the queryMap (such as during kd-tree construction), unmap the columns of the distances and neighbors matrices into neighborsOut and distancesOut, and also unmap the entries in each row of neighbors.

This is useful for the dual-tree case.

Parameters

neighbors	Matrix of neighbors resulting from neighbor search.
distances	Matrix of distances resulting from neighbor search.
referenceMap	Mapping of reference set to old points.
queryMap	Mapping of query set to old points.
neighborsOut	Matrix to store unmapped neighbors into.
distancesOut	Matrix to store unmapped distances into.
squareRoot	If true, take the square root of the distances.

20.19.3.2 void mlpack::neighbor::Unmap (const arma::Mat< size_t > & *neighbors*, const arma::mat & *distances*, const std::vector< size_t > & *referenceMap*, arma::Mat< size_t > & *neighborsOut*, arma::mat & *distancesOut*, const bool *squareRoot* = `false`)

Assuming that the datasets have been mapped using referenceMap (such as during kd-tree construction), unmap the columns of the distances and neighbors matrices into neighborsOut and distancesOut, and also unmap the entries in each row of neighbors.

This is useful for the single-tree case.

Parameters

neighbors	Matrix of neighbors resulting from neighbor search.
distances	Matrix of distances resulting from neighbor search.
referenceMap	Mapping of reference set to old points.
neighborsOut	Matrix to store unmapped neighbors into.
distancesOut	Matrix to store unmapped distances into.
squareRoot	If true, take the square root of the distances.

20.20 mlpack::nmf Namespace Reference

Classes

- class **HAlternatingLeastSquaresRule**

  The update rule for the encoding matrix H.
- class **HMultiplicativeDistanceRule**

  The update rule for the encoding matrix H.
- class **HMultiplicativeDivergenceRule**

  The update rule for the encoding matrix H.
- class **NMF**

  *This class implements the **NMF** (p. 353) on the given matrix V.*
- class **RandomAcolInitialization**

  *This class initializes the W matrix of the **NMF** (p. 353) algorithm by averaging p randomly chosen columns of V.*
- class **RandomInitialization**
- class **WAlternatingLeastSquaresRule**

  The update rule for the basis matrix W.
- class **WMultiplicativeDistanceRule**

  The update rule for the basis matrix W.
- class **WMultiplicativeDivergenceRule**

  The update rule for the basis matrix W.

20.21 mlpack::optimization Namespace Reference

Namespaces

- **test**

Classes

- class **AugLagrangian**

  *The **AugLagrangian** (p. 363) class implements the Augmented Lagrangian method of optimization.*
- class **AugLagrangianFunction**

  *This is a utility class used by **AugLagrangian** (p. 363), meant to wrap a LagrangianFunction into a function usable by a simple optimizer like L-BFGS.*
- class **AugLagrangianTestFunction**

  This function is taken from "Practical Mathematical Optimization" (Snyman), section 5.3.8 ("Application of the Augmented Lagrangian Method").

- class **GockenbachFunction**

  This function is taken from M.
- class **L_BFGS**

  The generic L-BFGS optimizer, which uses a back-tracking line search algorithm to minimize a function.
- class **LovaszThetaSDP**

  This function is the Lovasz-Theta semidefinite program, as implemented in the following paper:
- class **LRSDP**
- class **SGD**

  Stochastic Gradient Descent is a technique for minimizing a function which can be expressed as a sum of other functions.

20.22 mlpack::optimization::test Namespace Reference

Classes

- class **GeneralizedRosenbrockFunction**

  *The Generalized Rosenbrock function in n dimensions, defined by f(x) = sum_i^{n - 1} (f(i)(x)) f_i(x) = 100 * (x_i^2 - x_{i + 1})^2 + (1 - x_i)^2 x_0 = [-1.2, 1, -1.2, 1, ...].*
- class **RosenbrockFunction**

  The Rosenbrock function, defined by f(x) = f1(x) + f2(x) f1(x) = 100 (x2 - x1^2)^2 f2(x) = (1 - x1)^2 x_0 = [-1.2, 1].
- class **RosenbrockWoodFunction**

  The Generalized Rosenbrock function in 4 dimensions with the Wood Function in four dimensions.
- class **SGDTestFunction**

  Very, very simple test function which is the composite of three other functions.
- class **WoodFunction**

  The Wood function, defined by f(x) = f1(x) + f2(x) + f3(x) + f4(x) + f5(x) + f6(x) f1(x) = 100 (x2 - x1^2)^2 f2(x) = (1 - x1)^2 f3(x) = 90 (x4 - x3^2)^2 f4(x) = (1 - x3)^2 f5(x) = 10 (x2 + x4 - 2)^2 f6(x) = (1 / 10) (x2 - x4)^2 x_0 = [-3, -1, -3, -1].

20.23 mlpack::pca Namespace Reference

Classes

- class **PCA**

  *This class implements principal components analysis (**PCA** (p. 407)).*

20.24 mlpack::radical Namespace Reference

Classes

- class **Radical**

  An implementation of RADICAL, an algorithm for independent component analysis (ICA).

Functions

- void **WhitenFeatureMajorMatrix** (const arma::mat &matX, arma::mat &matXWhitened, arma::mat &matWhitening)

20.24.1 Function Documentation

20.24.1.1 void mlpack::radical::WhitenFeatureMajorMatrix (const arma::mat & *matX,* arma::mat & *matXWhitened,* arma::mat & *matWhitening*)

20.25 mlpack::range Namespace Reference

Range-search routines.

Classes

- class **RangeSearch**

  *The **RangeSearch** (p. 415) class is a template class for performing range searches.*
- class **RangeSearchRules**
- class **RangeSearchStat**

  *Statistic class for **RangeSearch** (p. 415), to be set to the StatisticType of the tree type that range search is being performed with.*

20.25.1 Detailed Description

Range-search routines.

20.26 mlpack::regression Namespace Reference

Regression methods.

Classes

- class **LARS**

  *An implementation of **LARS** (p. 429), a stage-wise homotopy-based algorithm for l1-regularized linear regression (LASSO) and l1+l2 regularized linear regression (Elastic Net).*
- class **LinearRegression**

  A simple linear regression algorithm using ordinary least squares.
- class **LogisticRegression**
- class **LogisticRegressionFunction**

  The log-likelihood function for the logistic regression objective function.

20.26.1 Detailed Description

Regression methods.

20.27 mlpack::sparse_coding Namespace Reference

Classes

- class **DataDependentRandomInitializer**

*A data-dependent random dictionary initializer for **SparseCoding** (p. 450).*

- class **NothingInitializer**

  *A DictionaryInitializer for **SparseCoding** (p. 450) which does not initialize anything; it is useful for when the dictionary is already known and will be set with **SparseCoding::Dictionary()** (p. 453).*

- class **RandomInitializer**

  *A DictionaryInitializer for use with the **SparseCoding** (p. 450) class.*

- class **SparseCoding**

  An implementation of Sparse Coding with Dictionary Learning that achieves sparsity via an l1-norm regularizer on the codes (LASSO) or an (l1+l2)-norm regularizer on the codes (the Elastic Net).

20.28 mlpack::tree Namespace Reference

Trees and tree-building procedures.

Classes

- class **BinarySpaceTree**

  A binary space partitioning tree, such as a KD-tree or a ball tree.

- class **CosineTree**
- class **CosineTreeBuilder**
- class **CoverTree**

  A cover tree is a tree specifically designed to speed up nearest-neighbor computation in high-dimensional spaces.

- struct **DualCoverTreeMapEntry**

  Forward declaration of struct to be used for traversal.

- class **EmptyStatistic**

  Empty statistic if you are not interested in storing statistics in your tree.

- class **FirstPointIsRoot**

  *This class is meant to be used as a choice for the policy class RootPointPolicy of the **CoverTree** (p. 491) class.*

- class **MRKDStatistic**

  Statistic for multi-resolution kd-trees.

- class **TreeTraits**

  *The **TreeTraits** (p. 521) class provides compile-time information on the characteristics of a given tree type.*

- class **TreeTraits**< **BinarySpaceTree**< **BoundType, StatisticType, MatType** > >

  *This is a specialization of the TreeType class to the **BinarySpaceTree** (p. 459) tree type.*

- class **TreeTraits**< **CoverTree**< **MetricType, RootPointPolicy, StatisticType** > >

  *The specialization of the **TreeTraits** (p. 521) class for the **CoverTree** (p. 491) tree type.*

20.28.1 Detailed Description

Trees and tree-building procedures. tree-building procedures.

Cosine Trees and building procedures.

20.29 mlpack::util Namespace Reference

Classes

- class **CLIDeleter**

  *Extremely simple class whose only job is to delete the existing **CLI** (p. 132) object at the end of execution.*
- class **NullOutStream**

  *Used for **Log::Debug** (p. 280) when not compiled with debugging symbols.*
- class **Option**

  *A static object whose constructor registers a parameter with the **CLI** (p. 132) class.*
- class **PrefixedOutStream**

  Allows us to output to an ostream with a prefix at the beginning of each line, in the same way we would output to cout or cerr.
- class **ProgramDoc**

  *A static object whose constructor registers program documentation with the **CLI** (p. 132) class.*
- class **SaveRestoreUtility**

Functions

- std::string **GetVersion** ()

  This will return either "mlpack x.y.z" or "mlpack trunk-rXXXXX" depending on whether or not this is a stable version of mlpack or an svn revision.
- std::string **Indent** (std::string input)

  A utility function that replaces all all newlines with a number of spaces depending on the indentation level.

Variables

- static **CLIDeleter cliDeleter**

  Declare the deleter.

20.29.1 Function Documentation

20.29.1.1 std::string mlpack::util::GetVersion ( )

This will return either "mlpack x.y.z" or "mlpack trunk-rXXXXX" depending on whether or not this is a stable version of mlpack or an svn revision.

20.29.1.2 std::string mlpack::util::Indent (std::string *input*)

A utility function that replaces all all newlines with a number of spaces depending on the indentation level.

20.29.2 Variable Documentation

20.29.2.1 CLIDeleter mlpack::util::cliDeleter [static]

Declare the deleter.

Definition at line 43 of file cli_deleter.hpp.

Chapter 21

Class Documentation

21.1 mlpack::bound::BallBound< VecType > Class Template Reference

Ball bound that works in the regular Euclidean metric space.

Public Types

- typedef VecType **Vec**

Public Member Functions

- **BallBound** ()
- **BallBound** (const size_t dimension)

  Create the ball bound with the specified dimensionality.
- **BallBound** (const double **radius**, const VecType &**center**)

  Create the ball bound with the specified radius and center.
- void **CalculateMidpoint** (VecType &centroid) const

  Gets the center.
- const VecType & **Center** () const

  Get the center point of the ball.
- VecType & **Center** ()

  Modify the center point of the ball.
- bool **Contains** (const VecType &point) const

  Determines if a point is within this bound.
- double **MaxDistance** (const VecType &point) const

  Computes maximum distance.
- double **MaxDistance** (const **BallBound** &other) const

  Computes maximum distance.
- double **MinDistance** (const VecType &point) const

  Calculates minimum bound-to-point squared distance.
- double **MinDistance** (const **BallBound** &other) const

  Calculates minimum bound-to-bound squared distance.
- **math::Range operator[]** (const size_t i) const

- const **BallBound** & **operator|=** (const **BallBound** &other)

  Expand the bound to include the given node.
- template<typename MatType >
  const **BallBound** & **operator|=** (const MatType &data)

  Expand the bound to include the given point.
- double **Radius** () const

  Get the radius of the ball.
- double & **Radius** ()

  Modify the radius of the ball.
- **math::Range RangeDistance** (const VecType &other) const

  Calculates minimum and maximum bound-to-point distance.
- **math::Range RangeDistance** (const **BallBound** &other) const

  Calculates minimum and maximum bound-to-bound distance.
- std::string **ToString** () const

  Returns a string representation of this object.

Private Attributes

- VecType **center**
- double **radius**

21.1.1 Detailed Description

template<typename VecType = arma::vec>class mlpack::bound::BallBound< VecType >

Ball bound that works in the regular Euclidean metric space.

Template Parameters

VecType	Type of vector (arma::vec or arma::spvec).

Definition at line 38 of file ballbound.hpp.

21.1.2 Member Typedef Documentation

21.1.2.1 template<typename VecType = arma::vec> typedef VecType mlpack::bound::BallBound< VecType >::Vec

Definition at line 41 of file ballbound.hpp.

21.1.3 Constructor & Destructor Documentation

21.1.3.1 template<typename VecType = arma::vec> mlpack::bound::BallBound< VecType >::BallBound ( ) [inline]

Definition at line 48 of file ballbound.hpp.

21.1.3.2 template<typename VecType = arma::vec> mlpack::bound::BallBound< VecType >::BallBound (const size_t *dimension*) [inline]

Create the ball bound with the specified dimensionality.

Parameters

dimension	Dimensionality of ball bound.

Definition at line 55 of file ballbound.hpp.

21.1.3.3 template<typename VecType = arma::vec> mlpack::bound::BallBound< VecType >::BallBound (const double *radius*, const VecType & *center*) [inline]

Create the ball bound with the specified radius and center.

Parameters

radius	Radius of ball bound.
center	Center of ball bound.

Definition at line 63 of file ballbound.hpp.

21.1.4 Member Function Documentation

21.1.4.1 template<typename VecType = arma::vec> void mlpack::bound::BallBound< VecType >::CalculateMidpoint (VecType & *centroid*) const

Gets the center.

Don't really use this directly. This is only here for consistency with DHrectBound, so it can plug in more directly if a "centroid" is needed.

21.1.4.2 template<typename VecType = arma::vec> const VecType& mlpack::bound::BallBound< VecType >::Center ( ) const [inline]

Get the center point of the ball.

Definition at line 72 of file ballbound.hpp.

References mlpack::bound::BallBound< VecType >::center.

21.1.4.3 template<typename VecType = arma::vec> VecType& mlpack::bound::BallBound< VecType >::Center ( ) [inline]

Modify the center point of the ball.

Definition at line 74 of file ballbound.hpp.

References mlpack::bound::BallBound< VecType >::center.

21.1.4.4 template<typename VecType = arma::vec> bool mlpack::bound::BallBound< VecType >::Contains (const VecType & *point*) const

Determines if a point is within this bound.

21.1.4.5 template<typename VecType = arma::vec> double mlpack::bound::BallBound< VecType >::MaxDistance (const VecType & *point*) const

Computes maximum distance.

21.1.4.6 template<typename VecType = arma::vec> double mlpack::bound::BallBound< VecType >::MaxDistance (const BallBound< VecType > & *other*) const

Computes maximum distance.

21.1.4.7 template<typename VecType = arma::vec> double mlpack::bound::BallBound< VecType >::MinDistance (const VecType & *point*) const

Calculates minimum bound-to-point squared distance.

21.1.4.8 template<typename VecType = arma::vec> double mlpack::bound::BallBound< VecType >::MinDistance (const BallBound< VecType > & *other*) const

Calculates minimum bound-to-bound squared distance.

21.1.4.9 template<typename VecType = arma::vec> math::Range mlpack::bound::BallBound< VecType >::operator[] (const size_t *i*) const

21.1.4.10 template<typename VecType = arma::vec> const BallBound& mlpack::bound::BallBound< VecType >::operator|= (const BallBound< VecType > & *other*)

Expand the bound to include the given node.

21.1.4.11 template<typename VecType = arma::vec> template<typename MatType > const BallBound& mlpack::bound::BallBound< VecType >::operator|= (const MatType & *data*)

Expand the bound to include the given point.

The centroid is recalculated to be the center of all of the given points.

Template Parameters

MatType	Type of matrix; could be arma::mat, arma::spmat, or a vector.
data	Data points to add.

21.1.4.12 template<typename VecType = arma::vec> double mlpack::bound::BallBound< VecType >::Radius ( ) const [inline]

Get the radius of the ball.

Definition at line 67 of file ballbound.hpp.

References mlpack::bound::BallBound< VecType >::radius.

21.1.4.13 template<typename VecType = arma::vec> double& mlpack::bound::BallBound< VecType >::Radius ( ) [inline]

Modify the radius of the ball.

Definition at line 69 of file ballbound.hpp.

References mlpack::bound::BallBound< VecType >::radius.

21.1.4.14 template<typename VecType = arma::vec> math::Range mlpack::bound::BallBound< VecType >::RangeDistance (const VecType & *other*) const

Calculates minimum and maximum bound-to-point distance.

21.1.4.15 template<typename VecType = arma::vec> math::Range mlpack::bound::BallBound< VecType >::RangeDistance (const BallBound< VecType > & *other*) const

Calculates minimum and maximum bound-to-bound distance.

Example: bound1.MinDistanceSq(other) for minimum distance.

21.1.4.16 template<typename VecType = arma::vec> std::string mlpack::bound::BallBound< VecType >::ToString ( ) const

Returns a string representation of this object.

21.1.5 Member Data Documentation

21.1.5.1 template<typename VecType = arma::vec> VecType mlpack::bound::BallBound< VecType >::center [private]

Definition at line 45 of file ballbound.hpp.

Referenced by mlpack::bound::BallBound< VecType >::Center().

21.1.5.2 template<typename VecType = arma::vec> double mlpack::bound::BallBound< VecType >::radius [private]

Definition at line 44 of file ballbound.hpp.

Referenced by mlpack::bound::BallBound< VecType >::Radius().

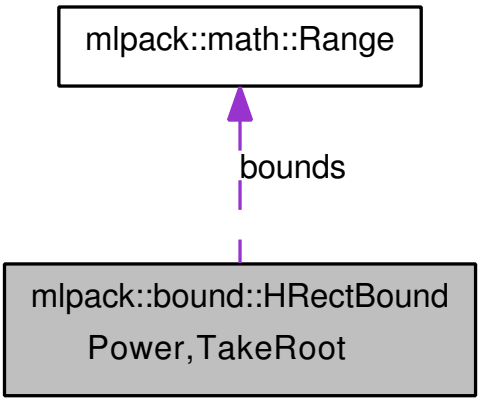
The documentation for this class was generated from the following file:

- src/mlpack/core/tree/**ballbound.hpp**

21.2 mlpack::bound::HRectBound< Power, TakeRoot > Class Template Reference

Hyper-rectangle bound for an L-metric.

Collaboration diagram for mlpack::bound::HRectBound< Power, TakeRoot >:



Public Types

- typedef **metric::LMetric**< Power, TakeRoot > **MetricType**

  This is the metric type that this bound is using.

Public Member Functions

- **HRectBound** ()

  Empty constructor; creates a bound of dimensionality 0.
- **HRectBound** (const size_t dimension)

  Initializes to specified dimensionality with each dimension the empty set.
- **HRectBound** (const **HRectBound** &other)

  Copy constructor; necessary to prevent memory leaks.
- ~**HRectBound** ()

  Destructor: clean up memory.
- void **Centroid** (arma::vec &centroid) const

  Calculates the centroid of the range, placing it into the given vector.
- void **Clear** ()

  Resets all dimensions to the empty set (so that this bound contains nothing).
- template<typename VecType >
  bool **Contains** (const VecType &point) const

  Determines if a point is within this bound.
- double **Diameter** () const

  Returns the diameter of the hyperrectangle (that is, the longest diagonal).
- size_t **Dim** () const

  Gets the dimensionality.
- template<typename VecType >
  double **MaxDistance** (const VecType &point) const

  Calculates maximum bound-to-point squared distance.
- double **MaxDistance** (const **HRectBound** &other) const

  Computes maximum distance.

- template<typename VecType >
  double **MinDistance** (const VecType &point) const

  Calculates minimum bound-to-point distance.
- double **MinDistance** (const **HRectBound** &other) const

  Calculates minimum bound-to-bound distance.
- **HRectBound** & **operator=** (const **HRectBound** &other)

  Same as copy constructor; necessary to prevent memory leaks.
- **math::Range** & **operator[]** (const size_t i)

  Get the range for a particular dimension. No bounds checking.
- const **math::Range** & **operator[]** (const size_t i) const

  Modify the range for a particular dimension. No bounds checking.
- template<typename MatType >
  HRectBound & **operator|=** (const MatType &data)

  Expands this region to include new points.
- **HRectBound** & **operator|=** (const **HRectBound** &other)

  Expands this region to encompass another bound.
- **math::Range RangeDistance** (const **HRectBound** &other) const

  Calculates minimum and maximum bound-to-bound distance.
- template<typename VecType >
  math::Range RangeDistance (const VecType &point) const

  Calculates minimum and maximum bound-to-point distance.
- std::string **ToString** () const

  Returns a string representation of this object.

Static Public Member Functions

- static **MetricType Metric** ()

  Return the metric associated with this bound.

Private Attributes

- **math::Range** * **bounds**

  The bounds for each dimension.
- size_t **dim**

  The dimensionality of the bound.

21.2.1 Detailed Description

template<int Power = 2, bool TakeRoot = true>class mlpack::bound::HRectBound< Power, TakeRoot >

Hyper-rectangle bound for an L-metric.

This should be used in conjunction with the LMetric class. Be sure to use the same template parameters for LMetric as you do for **HRectBound** (p. 115) – otherwise odd results may occur.

Template Parameters

Power	The metric to use; use 2 for Euclidean (L2).
TakeRoot	Whether or not the root should be taken (see LMetric documentation).

Definition at line 44 of file hrectbound.hpp.

21.2.2 Member Typedef Documentation

21.2.2.1 template<int Power = 2, bool TakeRoot = true> typedef metric::LMetric<Power, TakeRoot> mlpack::bound::HRectBound< Power, TakeRoot >::MetricType

This is the metric type that this bound is using.

Definition at line 48 of file hrectbound.hpp.

21.2.3 Constructor & Destructor Documentation

21.2.3.1 template<int Power = 2, bool TakeRoot = true> mlpack::bound::HRectBound< Power, TakeRoot >::HRectBound ( )

Empty constructor; creates a bound of dimensionality 0.

21.2.3.2 template<int Power = 2, bool TakeRoot = true> mlpack::bound::HRectBound< Power, TakeRoot >::HRectBound (const size_t *dimension*)

Initializes to specified dimensionality with each dimension the empty set.

21.2.3.3 template<int Power = 2, bool TakeRoot = true> mlpack::bound::HRectBound< Power, TakeRoot >::HRectBound (const HRectBound< Power, TakeRoot > & *other*)

Copy constructor; necessary to prevent memory leaks.

21.2.3.4 template<int Power = 2, bool TakeRoot = true> mlpack::bound::HRectBound< Power, TakeRoot >::~HRectBound ( )

Destructor: clean up memory.

21.2.4 Member Function Documentation

21.2.4.1 template<int Power = 2, bool TakeRoot = true> void mlpack::bound::HRectBound< Power, TakeRoot >::Centroid (arma::vec & *centroid*) const

Calculates the centroid of the range, placing it into the given vector.

Parameters

centroid	Vector which the centroid will be written to.

21.2.4.2 template<int Power = 2, bool TakeRoot = true> void mlpack::bound::HRectBound< Power, TakeRoot >::Clear ( )

Resets all dimensions to the empty set (so that this bound contains nothing).

21.2.4.3 template<int Power = 2, bool TakeRoot = true> template<typename VecType > bool mlpack::bound::HRectBound< Power, TakeRoot >::Contains (const VecType & *point*) const

Determines if a point is within this bound.

21.2.4.4 template<int Power = 2, bool TakeRoot = true> double mlpack::bound::HRectBound< Power, TakeRoot >::Diameter ( ) const

Returns the diameter of the hyperrectangle (that is, the longest diagonal).

21.2.4.5 template<int Power = 2, bool TakeRoot = true> size_t mlpack::bound::HRectBound< Power, TakeRoot >::Dim ( ) const [inline]

Gets the dimensionality.

Definition at line 76 of file hrectbound.hpp.

References mlpack::bound::HRectBound< Power, TakeRoot >::dim.

21.2.4.6 template<int Power = 2, bool TakeRoot = true> template<typename VecType > double mlpack::bound::HRectBound< Power, TakeRoot >::MaxDistance (const VecType & *point*) const

Calculates maximum bound-to-point squared distance.

Parameters

point	Point to which the maximum distance is requested.

21.2.4.7 template<int Power = 2, bool TakeRoot = true> double mlpack::bound::HRectBound< Power, TakeRoot >::MaxDistance (const HRectBound< Power, TakeRoot > & *other*) const

Computes maximum distance.

Parameters

other	Bound to which the maximum distance is requested.

21.2.4.8 template<int Power = 2, bool TakeRoot = true> static MetricType mlpack::bound::HRectBound< Power, TakeRoot >::Metric ( ) [inline], [static]

Return the metric associated with this bound.

Because it is an LMetric, it cannot store state, so we can make it on the fly. It is also static because the metric is only dependent on the template arguments.

Definition at line 173 of file hrectbound.hpp.

21.2.4.9 template<int Power = 2, bool TakeRoot = true> template<typename VecType > double mlpack::bound::HRectBound< Power, TakeRoot >::MinDistance (const VecType & *point*) const

Calculates minimum bound-to-point distance.

Parameters

point	Point to which the minimum distance is requested.

21.2.4.10 template<int Power = 2, bool TakeRoot = true> double mlpack::bound::HRectBound< Power, TakeRoot >::MinDistance (const HRectBound< Power, TakeRoot > & *other*) const

Calculates minimum bound-to-bound distance.

Parameters

other	Bound to which the minimum distance is requested.

21.2.4.11 template<int Power = 2, bool TakeRoot = true> HRectBound& mlpack::bound::HRectBound< Power, TakeRoot >::operator= (const HRectBound< Power, TakeRoot > & *other*)

Same as copy constructor; necessary to prevent memory leaks.

21.2.4.12 template<int Power = 2, bool TakeRoot = true> math::Range& mlpack::bound::HRectBound< Power, TakeRoot >::operator[] (const size_t *i*) [inline]

Get the range for a particular dimension. No bounds checking.

Definition at line 79 of file hrectbound.hpp.

References mlpack::bound::HRectBound< Power, TakeRoot >::bounds.

21.2.4.13 template<int Power = 2, bool TakeRoot = true> const math::Range& mlpack::bound::HRectBound< Power, TakeRoot >::operator[] (const size_t *i*) const [inline]

Modify the range for a particular dimension. No bounds checking.

Definition at line 81 of file hrectbound.hpp.

References mlpack::bound::HRectBound< Power, TakeRoot >::bounds.

21.2.4.14 template<int Power = 2, bool TakeRoot = true> template<typename MatType > HRectBound& mlpack::bound::HRectBound< Power, TakeRoot >::operator|= (const MatType & *data*)

Expands this region to include new points.

Template Parameters

MatType	Type of matrix; could be Mat, SpMat, a subview, or just a vector.

Parameters

data	Data points to expand this region to include.

21.2.4.15 template<int Power = 2, bool TakeRoot = true> HRectBound& mlpack::bound::HRectBound< Power, TakeRoot >::operator|= (const HRectBound< Power, TakeRoot > & *other*)

Expands this region to encompass another bound.

21.2.4.16 template<int Power = 2, bool TakeRoot = true> math::Range mlpack::bound::HRectBound< Power, TakeRoot >::RangeDistance (const HRectBound< Power, TakeRoot > & *other*) const

Calculates minimum and maximum bound-to-bound distance.

Parameters

other	Bound to which the minimum and maximum distances are requested.

21.2.4.17 template<int Power = 2, bool TakeRoot = true> template<typename VecType > math::Range mlpack::bound::HRectBound< Power, TakeRoot >::RangeDistance (const VecType & *point*) const

Calculates minimum and maximum bound-to-point distance.

Parameters

point	Point to which the minimum and maximum distances are requested.

21.2.4.18 template<int Power = 2, bool TakeRoot = true> std::string mlpack::bound::HRectBound< Power, TakeRoot >::ToString ( ) const

Returns a string representation of this object.

21.2.5 Member Data Documentation

21.2.5.1 template<int Power = 2, bool TakeRoot = true> math::Range* mlpack::bound::HRectBound< Power, TakeRoot >::bounds [private]

The bounds for each dimension.

Definition at line 179 of file hrectbound.hpp.

Referenced by mlpack::bound::HRectBound< Power, TakeRoot >::operator[]().

21.2.5.2 template<int Power = 2, bool TakeRoot = true> size_t mlpack::bound::HRectBound< Power, TakeRoot >::dim [private]

The dimensionality of the bound.

Definition at line 177 of file hrectbound.hpp.

Referenced by mlpack::bound::HRectBound< Power, TakeRoot >::Dim().

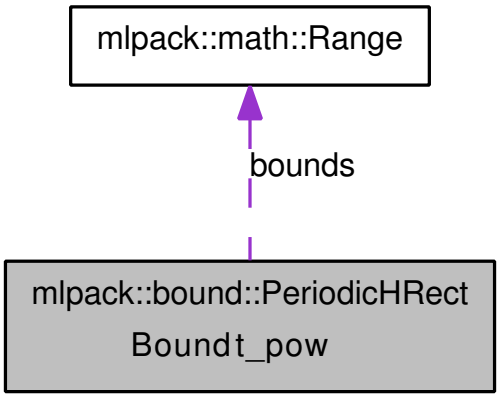
The documentation for this class was generated from the following file:

- src/mlpack/core/tree/**hrectbound.hpp**

21.3 mlpack::bound::PeriodicHRectBound< t_pow > Class Template Reference

Hyper-rectangle bound for an L-metric.

Collaboration diagram for mlpack::bound::PeriodicHRectBound< t_pow >:



Public Member Functions

- **PeriodicHRectBound** ()

  Empty constructor.
- **PeriodicHRectBound** (arma::vec **box**)

  Specifies the box size.
- **PeriodicHRectBound** (const **PeriodicHRectBound** &other)
- ~**PeriodicHRectBound** ()

  Destructor: clean up memory.
- const arma::vec & **Box** () const

  Returns the box vector.
- void **Centroid** (arma::vec &centroid) const
- void **Clear** ()

  Resets all dimensions to the empty set.
- bool **Contains** (const arma::vec &point) const

  Determines if a point is within this bound.
- size_t **Dim** () const

  Gets the dimensionality.
- double **MaxDistance** (const arma::vec &point) const

  Calculates maximum bound-to-point squared distance in the periodic bound case.
- double **MaxDistance** (const **PeriodicHRectBound** &other) const

Computes maximum bound-to-bound squared distance in the periodic bound case.

- double **MinDistance** (const arma::vec &point) const

  Calculates minimum bound-to-point squared distance in the periodic bound case.
- double **MinDistance** (const **PeriodicHRectBound** &other) const

  Calculates minimum bound-to-bound squared distance in the periodic bound case.
- **PeriodicHRectBound** & **operator=** (const **PeriodicHRectBound** &other)
- **math::Range** & **operator[]** (size_t i)

  Sets and gets the range for a particular dimension.
- const **math::Range operator[]** (size_t i) const
- **PeriodicHRectBound** & **operator|=** (const arma::vec &vector)

  Expands this region to include a new point.
- **PeriodicHRectBound** & **operator|=** (const **PeriodicHRectBound** &other)

  Expands this region to encompass another bound.
- **math::Range RangeDistance** (const arma::vec &point) const

  Calculates minimum and maximum bound-to-point squared distance in the periodic bound case.
- **math::Range RangeDistance** (const **PeriodicHRectBound** &other) const

  Calculates minimum and maximum bound-to-bound squared distance in the periodic bound case.
- void **SetBoxSize** (arma::vec **box**)

  Modifies the box to the desired dimenstions.
- std::string **ToString** () const

  Returns a string representation of an object.

Private Attributes

- **math::Range** * **bounds**
- arma::vec **box**
- size_t **dim**

21.3.1 Detailed Description

template<int t_pow = 2>class mlpack::bound::PeriodicHRectBound< t_pow >

Hyper-rectangle bound for an L-metric.

Template parameter t_pow is the metric to use; use 2 for Euclidean (L2).

Definition at line 38 of file periodichrectbound.hpp.

21.3.2 Constructor & Destructor Documentation

21.3.2.1 template<int t_pow = 2> mlpack::bound::PeriodicHRectBound< t_pow >::PeriodicHRectBound ( )

Empty constructor.

21.3.2.2 template<int t_pow = 2> mlpack::bound::PeriodicHRectBound< t_pow >::PeriodicHRectBound (arma::vec *box*)

Specifies the box size.

The dimensionality is set to the same of the box size, and the bounds are initialized to be empty.

21.3.2.3 template<int t_pow = 2> mlpack::bound::PeriodicHRectBound< t_pow >::PeriodicHRectBound (const PeriodicHRectBound< t_pow > & *other*)

21.3.2.4 template<int t_pow = 2> mlpack::bound::PeriodicHRectBound< t_pow >::~PeriodicHRectBound ( )

Destructor: clean up memory.

21.3.3 Member Function Documentation

21.3.3.1 template<int t_pow = 2> const arma::vec& mlpack::bound::PeriodicHRectBound< t_pow >::Box ( ) const [inline]

Returns the box vector.

Definition at line 72 of file periodichrectbound.hpp.

References mlpack::bound::PeriodicHRectBound< t_pow >::box.

21.3.3.2 template<int t_pow = 2> void mlpack::bound::PeriodicHRectBound< t_pow >::Centroid (arma::vec & *centroid*) const

21.3.3.3 template<int t_pow = 2> void mlpack::bound::PeriodicHRectBound< t_pow >::Clear ( )

Resets all dimensions to the empty set.

21.3.3.4 template<int t_pow = 2> bool mlpack::bound::PeriodicHRectBound< t_pow >::Contains (const arma::vec & *point*) const

Determines if a point is within this bound.

21.3.3.5 template<int t_pow = 2> size_t mlpack::bound::PeriodicHRectBound< t_pow >::Dim ( ) const [inline]

Gets the dimensionality.

Definition at line 80 of file periodichrectbound.hpp.

References mlpack::bound::PeriodicHRectBound< t_pow >::dim.

21.3.3.6 template<int t_pow = 2> double mlpack::bound::PeriodicHRectBound< t_pow >::MaxDistance (const arma::vec & *point*) const

Calculates maximum bound-to-point squared distance in the periodic bound case.

21.3.3.7 template<int t_pow = 2> double mlpack::bound::PeriodicHRectBound< t_pow >::MaxDistance (const PeriodicHRectBound< t_pow > & *other*) const

Computes maximum bound-to-bound squared distance in the periodic bound case.

21.3.3.8 template<int t_pow = 2> double mlpack::bound::PeriodicHRectBound< t_pow >::MinDistance (const arma::vec & *point*) const

Calculates minimum bound-to-point squared distance in the periodic bound case.

21.3.3.9 template<int t_pow = 2> double mlpack::bound::PeriodicHRectBound< t_pow >::MinDistance (const PeriodicHRectBound< t_pow > & *other*) const

Calculates minimum bound-to-bound squared distance in the periodic bound case.

Example: bound1.MinDistance(other) for minimum squared distance.

21.3.3.10 template<int t_pow = 2> PeriodicHRectBound& mlpack::bound::PeriodicHRectBound< t_pow >::operator= (const PeriodicHRectBound< t_pow > & *other*)

21.3.3.11 template<int t_pow = 2> math::Range& mlpack::bound::PeriodicHRectBound< t_pow >::operator[] (size_t *i*)

Sets and gets the range for a particular dimension.

21.3.3.12 template<int t_pow = 2> const math::Range mlpack::bound::PeriodicHRectBound< t_pow >::operator[] (size_t *i*) const

21.3.3.13 template<int t_pow = 2> PeriodicHRectBound& mlpack::bound::PeriodicHRectBound< t_pow >::operator|= (const arma::vec & *vector*)

Expands this region to include a new point.

21.3.3.14 template<int t_pow = 2> PeriodicHRectBound& mlpack::bound::PeriodicHRectBound< t_pow >::operator|= (const PeriodicHRectBound< t_pow > & *other*)

Expands this region to encompass another bound.

21.3.3.15 template<int t_pow = 2> math::Range mlpack::bound::PeriodicHRectBound< t_pow >::RangeDistance (const arma::vec & *point*) const

Calculates minimum and maximum bound-to-point squared distance in the periodic bound case.

21.3.3.16 template<int t_pow = 2> math::Range mlpack::bound::PeriodicHRectBound< t_pow >::RangeDistance (const PeriodicHRectBound< t_pow > & *other*) const

Calculates minimum and maximum bound-to-bound squared distance in the periodic bound case.

21.3.3.17 template<int t_pow = 2> void mlpack::bound::PeriodicHRectBound< t_pow >::SetBoxSize (arma::vec *box*)

Modifies the box to the desired dimenstions.

21.3.3.18 template<int t_pow = 2> std::string mlpack::bound::PeriodicHRectBound< t_pow >::ToString ( ) const

Returns a string representation of an object.

21.3.4 Member Data Documentation

21.3.4.1 template<int t_pow = 2> math::Range* mlpack::bound::PeriodicHRectBound< t_pow >::bounds [private]

Definition at line 155 of file periodichrectbound.hpp.

21.3.4.2 template<int t_pow = 2> arma::vec mlpack::bound::PeriodicHRectBound< t_pow >::box [private]

Definition at line 157 of file periodichrectbound.hpp.

Referenced by mlpack::bound::PeriodicHRectBound< t_pow >::Box().

21.3.4.3 template<int t_pow = 2> size_t mlpack::bound::PeriodicHRectBound< t_pow >::dim [private]

Definition at line 156 of file periodichrectbound.hpp.

Referenced by mlpack::bound::PeriodicHRectBound< t_pow >::Dim().

The documentation for this class was generated from the following file:

- src/mlpack/core/tree/**periodichrectbound.hpp**

21.4 mlpack::cf::CF Class Reference

This class implements Collaborative Filtering (**CF** (p. 126)).

Public Member Functions

- **CF** (const size_t **numRecs**, const size_t **numUsersForSimilarity**, arma::mat &**data**)

  *Create a **CF** (p. 126) object and (optionally) set the parameters with which collaborative filtering will be run.*
- **CF** (const size_t **numRecs**, arma::mat &**data**)

  *Create a **CF** (p. 126) object and (optionally) set the parameters which **CF** (p. 126) will be run with.*
- **CF** (arma::mat &**data**)

  *Create a **CF** (p. 126) object and (optionally) set the parameters which **CF** (p. 126) will be run with.*
- const arma::sp_mat & **CleanedData** () const

  Get the cleaned data matrix.
- const arma::mat & **Data** () const

  Get the data matrix.
- void **GetRecommendations** (arma::Mat< size_t > &recommendations)

  Generates default number of recommendations for all users.
- void **GetRecommendations** (arma::Mat< size_t > &recommendations, arma::Col< size_t > &users)

  Generates default number of recommendations for specified users.

- void **GetRecommendations** (arma::Mat< size_t > &recommendations, arma::Col< size_t > &users, size_t num)

  Generates a fixed number of recommendations for specified users.
- void **GetRecommendations** (arma::Mat< size_t > &recommendations, arma::Col< size_t > &users, size_t num, size_t neighbours)

  Generates a fixed number of recommendations for specified users.
- const arma::mat & **H** () const

  Get the Item Matrix.
- void **NumRecs** (size_t recs)

  Sets number of Recommendations.
- size_t **NumRecs** ()

  Gets numRecs.
- void **NumUsersForSimilarity** (size_t num)

  Sets number of user for calculating similarity.
- size_t **NumUsersForSimilarity** ()

  Gets number of users for calculating similarity/.
- const arma::mat & **Rating** () const

  Get the Rating Matrix.
- const arma::mat & **W** () const

  Get the User Matrix.

Private Member Functions

- void **CleanData** ()

  Converts the User, Item, Value Matrix to User-Item Table.
- void **InsertNeighbor** (const size_t queryIndex, const size_t pos, const size_t neighbor, const double value, arma::Mat< size_t > &recommendations, arma::mat &values) const

  Helper function to insert a point into the recommendation matrices.

Private Attributes

- arma::sp_mat **cleanedData**

  Cleaned data matrix.
- arma::mat **data**

  Initial data matrix.
- arma::mat **h**

  Item matrix.
- size_t **numRecs**

  Number of recommendations.
- size_t **numUsersForSimilarity**

  Number of users for similarity.
- arma::mat **rating**

  Rating matrix.
- arma::mat **w**

  User matrix.

21.4.1 Detailed Description

This class implements Collaborative Filtering (**CF** (p. 126)).

This implementation presently supports Alternating Least Squares (ALS) for collaborative filtering.

A simple example of how to run Collaborative Filtering is shown below.

```
extern arma::mat data; // (user, item, rating) table
extern arma::Col<size_t> users; // users seeking recommendations
arma::mat recommendations; // Recommendations
size_t numRecommendations = 10;

CF<> cf(data); // Default options.

// Generate the default number of recommendations for all users.
cf.GenerateRecommendations(recommendations);

// Generate the default number of recommendations for specified users.
cf.GenerateRecommendations(recommendations, users);

// Generate 10 recommendations for specified users.
cf.GenerateRecommendations(recommendations, users, numRecommendations);
```

The data matrix is a (user, item, rating) table. Each column in the matrix should have three rows. The first represents the user; the second represents the item; and the third represents the rating. The user and item, while they are in a matrix that holds doubles, should hold integer (or size_t) values.

Definition at line 68 of file cf.hpp.

21.4.2 Constructor & Destructor Documentation

21.4.2.1 mlpack::cf::CF::CF (const size_t *numRecs*, const size_t *numUsersForSimilarity*, arma::mat & *data*)

Create a **CF** (p. 126) object and (optionally) set the parameters with which collaborative filtering will be run.

Parameters

data	Initial (user,item,rating) matrix.
numRecs	Desired number of recommendations for each user.
numUsersForSimilarity	Size of the neighborhood.

21.4.2.2 mlpack::cf::CF::CF (const size_t *numRecs*, arma::mat & *data*)

Create a **CF** (p. 126) object and (optionally) set the parameters which **CF** (p. 126) will be run with.

Parameters

data	Initial User,Item,Rating Matrix
numRecs	Number of Recommendations for each user.

21.4.2.3 mlpack::cf::CF::CF (arma::mat & *data*)

Create a **CF** (p. 126) object and (optionally) set the parameters which **CF** (p. 126) will be run with.

Parameters

data	Initial User,Item,Rating Matrix

21.4.3 Member Function Documentation

21.4.3.1 void mlpack::cf::CF::CleanData ( ) [private]

Converts the User, Item, Value Matrix to User-Item Table.

21.4.3.2 const arma::sp_mat& mlpack::cf::CF::CleanedData ( ) const [inline]

Get the cleaned data matrix.

Definition at line 144 of file cf.hpp.

References cleanedData.

21.4.3.3 const arma::mat& mlpack::cf::CF::Data ( ) const [inline]

Get the data matrix.

Definition at line 142 of file cf.hpp.

References data.

21.4.3.4 void mlpack::cf::CF::GetRecommendations (arma::Mat< size_t > & *recommendations*)

Generates default number of recommendations for all users.

Parameters

recommendations	Matrix to save recommendations into.

21.4.3.5 void mlpack::cf::CF::GetRecommendations (arma::Mat< size_t > & *recommendations,* arma::Col< size_t > & *users*)

Generates default number of recommendations for specified users.

Parameters

recommendations	Matrix to save recommendations
users	Users for which recommendations are to be generated

21.4.3.6 void mlpack::cf::CF::GetRecommendations (arma::Mat< size_t > & *recommendations,* arma::Col< size_t > & *users,* size_t *num*)

Generates a fixed number of recommendations for specified users.

Parameters

recommendations	Matrix to save recommendations
users	Users for which recommendations are to be generated
num	Number of Recommendations

21.4.3.7 void mlpack::cf::CF::GetRecommendations (arma::Mat< size_t > & *recommendations*, arma::Col< size_t > & *users*, size_t *num*, size_t *neighbours*)

Generates a fixed number of recommendations for specified users.

Parameters

recommendations	Matrix to save recommendations
users	Users for which recommendations are to be generated
num	Number of Recommendations
neighbours	Number of user to be considered while calculating the neighbourhood

21.4.3.8 const arma::mat& mlpack::cf::CF::H ( ) const [inline]

Get the Item Matrix.

Definition at line 138 of file cf.hpp.

References h.

21.4.3.9 void mlpack::cf::CF::InsertNeighbor (const size_t *queryIndex*, const size_t *pos*, const size_t *neighbor*, const double *value*, arma::Mat< size_t > & *recommendations*, arma::mat & *values*) const [private]

Helper function to insert a point into the recommendation matrices.

Parameters

queryIndex	Index of point whose recommendations we are inserting into.
pos	Position in list to insert into.
neighbor	Index of item being inserted as a recommendation.
value	Value of recommendation.

21.4.3.10 void mlpack::cf::CF::NumRecs (size_t *recs*) [inline]

Sets number of Recommendations.

Definition at line 100 of file cf.hpp.

References mlpack::Log::Warn.

21.4.3.11 size_t mlpack::cf::CF::NumRecs ( ) [inline]

Gets numRecs.

Definition at line 112 of file cf.hpp.

References numRecs.

21.4.3.12 void mlpack::cf::CF::NumUsersForSimilarity (size_t *num*) [inline]

Sets number of user for calculating similarity.

Definition at line 118 of file cf.hpp.

References mlpack::Log::Warn.

21.4.3.13 size_t mlpack::cf::CF::NumUsersForSimilarity ( ) [inline]

Gets number of users for calculating similarity/.

Definition at line 130 of file cf.hpp.

References numUsersForSimilarity.

21.4.3.14 const arma::mat& mlpack::cf::CF::Rating ( ) const [inline]

Get the Rating Matrix.

Definition at line 140 of file cf.hpp.

References rating.

21.4.3.15 const arma::mat& mlpack::cf::CF::W ( ) const [inline]

Get the User Matrix.

Definition at line 136 of file cf.hpp.

References w.

21.4.4 Member Data Documentation

21.4.4.1 arma::sp_mat mlpack::cf::CF::cleanedData [private]

Cleaned data matrix.

Definition at line 199 of file cf.hpp.

Referenced by CleanedData().

21.4.4.2 arma::mat mlpack::cf::CF::data [private]

Initial data matrix.

Definition at line 197 of file cf.hpp.

Referenced by Data().

21.4.4.3 arma::mat mlpack::cf::CF::h [private]

Item matrix.

Definition at line 193 of file cf.hpp.

Referenced by H().

21.4.4.4 size_t mlpack::cf::CF::numRecs [private]

Number of recommendations.

Definition at line 187 of file cf.hpp.

Referenced by NumRecs().

21.4.4.5 size_t mlpack::cf::CF::numUsersForSimilarity [private]

Number of users for similarity.

Definition at line 189 of file cf.hpp.

Referenced by NumUsersForSimilarity().

21.4.4.6 arma::mat mlpack::cf::CF::rating [private]

Rating matrix.

Definition at line 195 of file cf.hpp.

Referenced by Rating().

21.4.4.7 arma::mat mlpack::cf::CF::w [private]

User matrix.

Definition at line 191 of file cf.hpp.

Referenced by W().

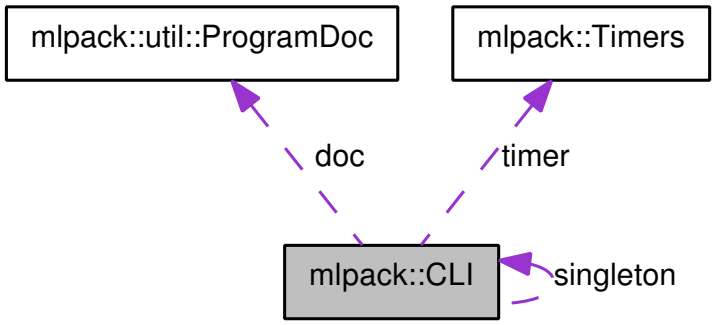
The documentation for this class was generated from the following file:

- src/mlpack/methods/cf/**cf.hpp**

21.5 mlpack::CLI Class Reference

Parses the command line for parameters and holds user-specified parameters.

Collaboration diagram for mlpack::CLI:



Public Member Functions

- ∼**CLI** ()

  Destructor.

Static Public Member Functions

- static void **Add** (const std::string &path, const std::string &description, const std::string &alias="", bool required=false)

  Adds a parameter to the hierarchy; use the PARAM_∗() macros instead of this (i.e.
- template<class T >
  static void **Add** (const std::string &identifier, const std::string &description, const std::string &alias="", bool required=false)

  Adds a parameter to the hierarchy; use the PARAM_∗() macros instead of this (i.e.
- static void **AddFlag** (const std::string &identifier, const std::string &description, const std::string &alias="")

  *Adds a flag parameter to the hierarchy; use **PARAM_FLAG()** (p. 626) instead of this.*
- static void **DefaultMessages** ()

  Parses the parameters for 'help' and 'info'.
- static void **Destroy** ()

  *Destroy the **CLI** (p. 132) object.*
- static std::string **GetDescription** (const std::string &identifier)

  Get the description of the specified node.
- template<typename T >
  static T & **GetParam** (const std::string &identifier)

  Grab the value of type T found while parsing.
- static **CLI** & **GetSingleton** ()

  Retrieve the singleton.
- static bool **HasParam** (const std::string &identifier)

  See if the specified flag was found while parsing.
- static std::string **HyphenateString** (const std::string &str, int padding)

  Hyphenate a string or split it onto multiple 80-character lines, with some amount of padding on each line.
- static void **ParseCommandLine** (int argc, char ∗∗argv)

  Parses the commandline for arguments.
- static void **ParseStream** (std::istream &stream)

Parses a stream for arguments.

- static void **Print** ()

  Print out the current hierarchy.

- static void **PrintHelp** (const std::string &param="")

  Print out the help info of the hierarchy.

- static void **RegisterProgramDoc** (**util::ProgramDoc** ∗**doc**)

  Registers a ProgramDoc object, which contains documentation about the program.

- static void **RemoveDuplicateFlags** (po::basic_parsed_options< char > &bpo)

  Removes duplicate flags.

Public Attributes

- **util::ProgramDoc** ∗ **doc**

  Pointer to the ProgramDoc object.

Private Types

- typedef std::map< std::string, std::string > **amap_t**

  Map for aliases, from alias to actual name.

- typedef std::map< std::string, **ParamData** > **gmap_t**

  Map of global values.

Private Member Functions

- **CLI** ()

  Make the constructor private, to preclude unauthorized instances.

- **CLI** (const std::string &optionsName)

  Initialize desc with a particular name.

- **CLI** (const **CLI** &other)

  Private copy constructor; we don't want copies floating around.

Static Private Member Functions

- static void **AddAlias** (const std::string &alias, const std::string &original)

  Maps a given alias to a given parameter.

- static std::string **AliasReverseLookup** (const std::string &value)

  Returns an alias, if given the name of the original.

- static void **RequiredOptions** ()

  Checks that all required parameters have been specified on the command line.

- static std::string **SanitizeString** (const std::string &str)

  Cleans up input pathnames, rendering strings such as /foo/bar and foo/bar/ equivalent inputs.

- static void **UpdateGmap** ()

  Parses the values given on the command line, overriding any default values.

Private Attributes

- **amap_t aliasValues**
- po::options_description **desc**

  The documentation and names of options.
- bool **didParse**

  *True, if **CLI** (p. 132) was used to parse command line options.*
- **gmap_t globalValues**
- std::string **programName**

  Hold the name of the program for –version.
- std::list< std::string > **requiredOptions**

  Pathnames of required options.
- **Timers timer**

  Holds the timer objects.
- po::variables_map **vmap**

  Values of the options given by user.

Static Private Attributes

- static **CLI** * **singleton**

  The singleton itself.

Friends

- class **Timer**

  *So that **Timer::Start()** (p. 457) and **Timer::Stop()** (p. 457) can access the timer variable.*

21.5.1 Detailed Description

Parses the command line for parameters and holds user-specified parameters.

The **CLI** (p. 132) class is a subsystem by which parameters for machine learning methods can be specified and accessed. In conjunction with the macros PARAM_DOUBLE, PARAM_INT, PARAM_STRING, PARAM_FLAG, and others, this class aims to make user configurability of MLPACK methods very easy. There are only three methods in **CLI** (p. 132) that a user should need: **CLI::ParseCommandLine()** (p. 142), **CLI::GetParam()** (p. 141), and **CLI::HasParam()** (p. 142) (in addition to the PARAM_*() macros).

21.5.2 Adding parameters to a program

```
$ ./executable --bar=5
```

Note

The = is optional; a space can also be used.

A parameter is specified by using one of the following macros (this is not a complete list; see core/io/cli.hpp):

- **PARAM_FLAG(ID, DESC, ALIAS)** (p. 626)

- **PARAM_DOUBLE(ID, DESC, ALIAS, DEF)** (p. 625)
- **PARAM_INT(ID, DESC, ALIAS, DEF)** (p. 627)
- **PARAM_STRING(ID, DESC, ALIAS, DEF)** (p. 628)

Parameters

ID	Name of the parameter.
DESC	Short description of the parameter (one/two sentences).
ALIAS	An alias for the parameter.
DEF	Default value of the parameter.

The flag (boolean) type automatically defaults to false; it is specified merely as a flag on the command line (no '=true' is required).

Here is an example of a few parameters being defined; this is for the AllkNN executable (methods/neighbor_search/allknn_main.cpp):

```
PARAM_STRING_REQ("reference_file", "File containing the reference dataset.",
    "r");
PARAM_STRING_REQ("distances_file", "File to output distances into.", "d");
PARAM_STRING_REQ("neighbors_file", "File to output neighbors into.", "n");
PARAM_INT_REQ("k", "Number of furthest neighbors to find.", "k");
PARAM_STRING("query_file", "File containing query points (optional).", "q",
    "");
PARAM_INT("leaf_size", "Leaf size for tree building.", "l", 20);
PARAM_FLAG("naive", "If true, O(n^2) naive mode is used for computation.",
    "N");
PARAM_FLAG("single_mode", "If true, single-tree search is used (as opposed "
    "to dual-tree search.", "s");
```

More documentation is available on the PARAM_∗() macros in the documentation for core/io/cli.hpp.

21.5.3 Documenting the program itself

In addition to allowing documentation for each individual parameter and module, the **PROGRAM_INFO()** (p. 630) macro provides support for documenting the program itself. There should only be one instance of the **PROGRAM_INFO()** (p. 630) macro. Below is an example:

```
PROGRAM_INFO("Maximum Variance Unfolding", "This program performs maximum "
   "variance unfolding on the given dataset, writing a lower-dimensional "
   "unfolded dataset to the given output file.");
```

This description should be verbose, and explain to a non-expert user what the program does and how to use it. If relevant, paper citations should be included.

21.5.4 Parsing the command line with CLI

To have **CLI** (p. 132) parse the command line at the beginning of code execution, only a call to **ParseCommandLine()** (p. 142) is necessary:

```
int main(int argc, char** argv)
{
  CLI::ParseCommandLine(argc, argv);

  ...
}
```

CLI (p. 132) provides –help and –info options which give nicely formatted documentation of each option; the documentation is generated from the DESC arguments in the PARAM_∗() macros.

21.5.5 Getting parameters with CLI

When the parameters have been defined, the next important thing is how to access them. For this, the **HasParam()** (p. 142) and **GetParam()** (p. 141) methods are used. For instance, to see if the user passed the flag (boolean) "naive":

```
if (CLI::HasParam("naive"))
{
  Log::Info << "Naive has been passed!" << std::endl;
}
```

To get the value of a parameter, such as a string, use GetParam:

```
const std::string filename = CLI::GetParam<std::string>("filename");
```

Note

Options should only be defined in files which define `main()` (that is, main executables). If options are defined elsewhere, they may be spuriously included into other executables and confuse users. Similarly, if your executable has options which you did not define, it is probably because the option is defined somewhere else and included in your executable.

Bug The **COUNTER** variable is used in most cases to guarantee a unique global identifier for options declared using the PARAM_∗() macros. However, not all compilers have this support–most notably, gcc $<$ 4.3. In that case, the **LINE** macro is used as an attempt to get a unique global identifier, but collisions are still possible, and they produce bizarre error messages. See `http://mlpack.org/trac/ticket/74` for more information.

Definition at line 531 of file cli.hpp.

21.5.6 Member Typedef Documentation

21.5.6.1 typedef std::map<std::string, std::string> mlpack::CLI::amap_t [private]

Map for aliases, from alias to actual name.

Definition at line 699 of file cli.hpp.

21.5.6.2 typedef std::map<std::string, ParamData> mlpack::CLI::gmap_t [private]

Map of global values.

Definition at line 695 of file cli.hpp.

21.5.7 Constructor & Destructor Documentation

21.5.7.1 mlpack::CLI::∼CLI ( )

Destructor.

21.5.7.2 mlpack::CLI::CLI ( ) [private]

Make the constructor private, to preclude unauthorized instances.

21.5.7.3 mlpack::CLI::CLI (const std::string & *optionsName*) [private]

Initialize desc with a particular name.

Parameters

optionsName	Name of the module, as far as boost is concerned.

21.5.7.4 mlpack::CLI::CLI (const CLI & *other*) [private]

Private copy constructor; we don't want copies floating around.

21.5.8 Member Function Documentation

21.5.8.1 static void mlpack::CLI::Add (const std::string & *path,* const std::string & *description,* const std::string & *alias* = " ", bool *required* = false) [static]

Adds a parameter to the hierarchy; use the PARAM_∗() macros instead of this (i.e.

PARAM_INT() (p. 627)). Uses char∗ and not std::string since the vast majority of use cases will be literal strings.

Parameters

identifier	The name of the parameter.
description	Short string description of the parameter.
alias	An alias for the parameter, defaults to "" which is no alias. ("").
required	Indicates if parameter must be set on command line.

21.5.8.2 template<class T > static void mlpack::CLI::Add (const std::string & *identifier,* const std::string & *description,* const std::string & *alias* = " ", bool *required* = false) [static]

Adds a parameter to the hierarchy; use the PARAM_∗() macros instead of this (i.e.

PARAM_INT() (p. 627)). Uses char∗ and not std::string since the vast majority of use cases will be literal strings. If the argument requires a parameter, you must specify a type.

Parameters

identifier	The name of the parameter.
description	Short string description of the parameter.
alias	An alias for the parameter, defaults to "" which is no alias.
required	Indicates if parameter must be set on command line.

21.5.8.3 static void mlpack::CLI::AddAlias (const std::string & *alias,* const std::string & *original*) [static], [private]

Maps a given alias to a given parameter.

Parameters

alias	The name of the alias to be mapped.
original	The name of the parameter to be mapped.

21.5.8.4 static void mlpack::CLI::AddFlag (const std::string & *identifier,* const std::string & *description,* const std::string & *alias =* "") [static]

Adds a flag parameter to the hierarchy; use **PARAM_FLAG()** (p. 626) instead of this.

Parameters

identifier	The name of the paramater.
description	Short string description of the parameter.
alias	An alias for the parameter, defaults to "" which is no alias.

21.5.8.5 static std::string mlpack::CLI::AliasReverseLookup (const std::string & *value*) [static],[private]

Returns an alias, if given the name of the original.

Parameters

value	The value in a key:value pair where the key is an alias.

Returns

The alias associated with value.

21.5.8.6 static void mlpack::CLI::DefaultMessages ( ) [static]

Parses the parameters for 'help' and 'info'.

If found, will print out the appropriate information and kill the program.

21.5.8.7 static void mlpack::CLI::Destroy ( ) [static]

Destroy the **CLI** (p. 132) object.

This resets the pointer to the singleton, so in case someone tries to access it after destruction, a new one will be made (the program will not fail).

21.5.8.8 static std::string mlpack::CLI::GetDescription (const std::string & *identifier*) [static]

Get the description of the specified node.

Parameters

identifier	Name of the node in question.

Returns

Description of the node in question.

21.5.8.9 template<typename T > static T& mlpack::CLI::GetParam (const std::string & *identifier*) [static]

Grab the value of type T found while parsing.

You can set the value using this reference safely.

Parameters

identifier	The name of the parameter in question.

21.5.8.10 static CLI& mlpack::CLI::GetSingleton ( ) [static]

Retrieve the singleton.

Not exposed to the outside, so as to spare users some ungainly x.GetSingleton().foo() syntax.

In this case, the singleton is used to store data for the static methods, as there is no point in defining static methods only to have users call private instance methods.

Returns

The singleton instance for use in the static methods.

21.5.8.11 static bool mlpack::CLI::HasParam (const std::string & *identifier*) [static]

See if the specified flag was found while parsing.

Parameters

identifier	The name of the parameter in question.

21.5.8.12 static std::string mlpack::CLI::HyphenateString (const std::string & *str,* int *padding*) [static]

Hyphenate a string or split it onto multiple 80-character lines, with some amount of padding on each line.

This is ued for option output.

Parameters

str	String to hyphenate (splits are on ' ').
padding	Amount of padding on the left for each new line.

21.5.8.13 static void mlpack::CLI::ParseCommandLine (int *argc,* char ** *argv*) [static]

Parses the commandline for arguments.

Parameters

argc	The number of arguments on the commandline.
argv	The array of arguments as strings.

21.5.8.14 static void mlpack::CLI::ParseStream (std::istream & *stream*) [static]

Parses a stream for arguments.

Parameters

stream	The stream to be parsed.

21.5.8.15 static void mlpack::CLI::Print ( ) [static]

Print out the current hierarchy.

21.5.8.16 static void mlpack::CLI::PrintHelp (const std::string & *param* = " ") [static]

Print out the help info of the hierarchy.

21.5.8.17 static void mlpack::CLI::RegisterProgramDoc (util::ProgramDoc * *doc*) [static]

Registers a ProgramDoc object, which contains documentation about the program.

If this method has been called before (that is, if two ProgramDocs are instantiated in the program), a fatal error will occur.

Parameters

doc	Pointer to the ProgramDoc object.

21.5.8.18 static void mlpack::CLI::RemoveDuplicateFlags (po::basic_parsed_options< char > & *bpo*) [static]

Removes duplicate flags.

Parameters

bpo	The basic_program_options to remove duplicate flags from.

21.5.8.19 static void mlpack::CLI::RequiredOptions ( ) [static], [private]

Checks that all required parameters have been specified on the command line.

If any have not been specified, an error message is printed and the program is terminated.

21.5.8.20 static std::string mlpack::CLI::SanitizeString (const std::string & *str*) [static], [private]

Cleans up input pathnames, rendering strings such as /foo/bar and foo/bar/ equivalent inputs.

Parameters

str	Input string.

Returns

Sanitized string.

21.5.8.21 static void mlpack::CLI::UpdateGmap ( ) [static], [private]

Parses the values given on the command line, overriding any default values.

21.5.9 Friends And Related Function Documentation

21.5.9.1 friend class Timer [friend]

So that **Timer::Start()** (p. 457) and **Timer::Stop()** (p. 457) can access the timer variable.

Definition at line 715 of file cli.hpp.

21.5.10 Member Data Documentation

21.5.10.1 amap_t mlpack::CLI::aliasValues [private]

Definition at line 700 of file cli.hpp.

21.5.10.2 po::options_description mlpack::CLI::desc [private]

The documentation and names of options.

Definition at line 686 of file cli.hpp.

21.5.10.3 bool mlpack::CLI::didParse [private]

True, if **CLI** (p. 132) was used to parse command line options.

Definition at line 706 of file cli.hpp.

21.5.10.4 util::ProgramDoc* mlpack::CLI::doc

Pointer to the ProgramDoc object.

Definition at line 719 of file cli.hpp.

21.5.10.5 gmap_t mlpack::CLI::globalValues [private]

Definition at line 696 of file cli.hpp.

21.5.10.6 std::string mlpack::CLI::programName [private]

Hold the name of the program for –version.

Definition at line 709 of file cli.hpp.

21.5.10.7 std::list<std::string> mlpack::CLI::requiredOptions [private]

Pathnames of required options.

Definition at line 692 of file cli.hpp.

21.5.10.8 CLI* mlpack::CLI::singleton [static], [private]

The singleton itself.

Definition at line 703 of file cli.hpp.

21.5.10.9 Timers mlpack::CLI::timer [private]

Holds the timer objects.

Definition at line 712 of file cli.hpp.

21.5.10.10 po::variables_map mlpack::CLI::vmap [private]

Values of the options given by user.

Definition at line 689 of file cli.hpp.

The documentation for this class was generated from the following file:

- src/mlpack/core/util/**cli.hpp**

21.6 mlpack::det::DTree Class Reference

A density estimation tree is similar to both a decision tree and a space partitioning tree (like a kd-tree).

Collaboration diagram for mlpack::det::DTree:

Public Member Functions

- **DTree** ()

  Create an empty density estimation tree.
- **DTree** (const arma::vec &**maxVals**, const arma::vec &**minVals**, const size_t totalPoints)

  Create a density estimation tree with the given bounds and the given number of total points.
- **DTree** (arma::mat &data)

  Create a density estimation tree on the given data.
- **DTree** (const arma::vec &**maxVals**, const arma::vec &**minVals**, const size_t **start**, const size_t **end**, const double **logNegError**)

  Create a child node of a density estimation tree given the bounding box specified by maxVals and minVals, using the size given in start and end and the specified error.
- **DTree** (const arma::vec &**maxVals**, const arma::vec &**minVals**, const size_t totalPoints, const size_t **start**, const size_t **end**)

  Create a child node of a density estimation tree given the bounding box specified by maxVals and minVals, using the size given in start and end, and calculating the error with the total number of points given.
- ~**DTree** ()

Clean up memory allocated by the tree.

- double **AlphaUpper** () const

  Return the upper part of the alpha sum.

- double **ComputeValue** (const arma::vec &query) const

  Compute the logarithm of the density estimate of a given query point.

- void **ComputeVariableImportance** (arma::vec &importances) const

  Compute the variable importance of each dimension in the learned tree.

- size_t **End** () const

  Return the first index of a point not contained in this node.

- int **FindBucket** (const arma::vec &query) const

  Return the tag of the leaf containing the query.

- double **Grow** (arma::mat &data, arma::Col< size_t > &oldFromNew, const bool useVolReg=false, const size_t maxLeafSize=10, const size_t minLeafSize=5)

  Greedily expand the tree.

- **DTree** * **Left** () const

  Return the left child.

- double **LogNegativeError** (const size_t totalPoints) const

  Compute the log-negative-error for this point, given the total number of points in the dataset.

- double **LogNegError** () const

  Return the log negative error of this node.

- double **LogVolume** () const

  Return the inverse of the volume of this node.

- const arma::vec & **MaxVals** () const

  Return the maximum values.

- arma::vec & **MaxVals** ()

  Modify the maximum values.

- const arma::vec & **MinVals** () const

  Return the minimum values.

- arma::vec & **MinVals** ()

  Modify the minimum values.

- double **PruneAndUpdate** (const double oldAlpha, const size_t points, const bool useVolReg=false)

  Perform alpha pruning on a tree.

- double **Ratio** () const

  Return the ratio of points in this node to the points in the whole dataset.

- **DTree** * **Right** () const

  Return the right child.

- bool **Root** () const

  Return whether or not this is the root of the tree.

- size_t **SplitDim** () const

  Return the split dimension of this node.

- double **SplitValue** () const

  Return the split value of this node.

- size_t **Start** () const

  Return the starting index of points contained in this node.

- size_t **SubtreeLeaves** () const

  Return the number of leaves which are descendants of this node.

- double **SubtreeLeavesLogNegError** () const

Return the log negative error of all descendants of this node.

- int **TagTree** (const int tag=0)

  Index the buckets for possible usage later; this results in every leaf in the tree having a specific tag (accessible with BucketTag()).

- bool **WithinRange** (const arma::vec &query) const

  Return whether a query point is within the range of this node.

- void **WriteTree** (FILE *fp, const size_t level=0) const

  Print the tree in a depth-first manner (this function is called recursively).

Private Member Functions

- bool **FindSplit** (const arma::mat &data, size_t &**splitDim**, double &**splitValue**, double &leftError, double &rightError, const size_t maxLeafSize=10, const size_t minLeafSize=5) const

  Find the dimension to split on.

- size_t **SplitData** (arma::mat &data, const size_t **splitDim**, const double **splitValue**, arma::Col< size_t > &oldFromNew) const

  Split the data, returning the number of points left of the split.

Private Attributes

- double **alphaUpper**

  Upper part of alpha sum; used for pruning.

- int **bucketTag**

  The tag for the leaf, used for hashing points.

- size_t **end**

  The index of the last point in the dataset contained in this node (and its children).

- **DTree** * **left**

  The left child.

- double **logNegError**

  log-negative-L2-error of the node.

- double **logVolume**

  The logarithm of the volume of the node.

- arma::vec **maxVals**

  Upper half of bounding box for this node.

- arma::vec **minVals**

  Lower half of bounding box for this node.

- double **ratio**

  Ratio of the number of points in the node to the total number of points.

- **DTree** * **right**

  The right child.

- bool **root**

  If true, this node is the root of the tree.

- size_t **splitDim**

  The splitting dimension for this node.

- double **splitValue**

  The split value on the splitting dimension for this node.

- size_t **start**

The index of the first point in the dataset contained in this node (and its children).

- size_t **subtreeLeaves**

  Number of leaves of the subtree.

- double **subtreeLeavesLogNegError**

  Sum of the error of the leaves of the subtree.

21.6.1 Detailed Description

A density estimation tree is similar to both a decision tree and a space partitioning tree (like a kd-tree).

Each leaf represents a constant-density hyper-rectangle. The tree is constructed in such a way as to minimize the integrated square error between the probability distribution of the tree and the observed probability distribution of the data. Because the tree is similar to a decision tree, the density estimation tree can provide very fast density estimates for a given point.

For more information, see the following paper:

```
@incollection{ram2011,
  author = {Ram, Parikshit and Gray, Alexander G.},
  title = {Density estimation trees},
  booktitle = {{Proceedings of the 17th ACM SIGKDD International Conference
      on Knowledge Discovery and Data Mining}},
  series = {KDD '11},
  year = {2011},
  pages = {627--635}
}
```

Definition at line 54 of file dtree.hpp.

21.6.2 Constructor & Destructor Documentation

21.6.2.1 mlpack::det::DTree::DTree ( )

Create an empty density estimation tree.

21.6.2.2 mlpack::det::DTree::DTree (const arma::vec & *maxVals*, const arma::vec & *minVals*, const size_t *totalPoints*)

Create a density estimation tree with the given bounds and the given number of total points.

Children will not be created.

Parameters

maxVals	Maximum values of the bounding box.
minVals	Minimum values of the bounding box.
totalPoints	Total number of points in the dataset.

21.6.2.3 mlpack::det::DTree::DTree (arma::mat & *data*)

Create a density estimation tree on the given data.

Children will be created following the procedure outlined in the paper. The data will be modified; it will be reordered similar to the way BinarySpaceTree modifies datasets.

Parameters

data	Dataset to build tree on.

21.6.2.4 mlpack::det::DTree::DTree (const arma::vec & *maxVals,* const arma::vec & *minVals,* const size_t *start,* const size_t *end,* const double *logNegError*)

Create a child node of a density estimation tree given the bounding box specified by maxVals and minVals, using the size given in start and end and the specified error.

Children of this node will not be created recursively.

Parameters

maxVals	Upper bound of bounding box.
minVals	Lower bound of bounding box.
start	Start of points represented by this node in the data matrix.
end	End of points represented by this node in the data matrix.
error	log-negative error of this node.

21.6.2.5 mlpack::det::DTree::DTree (const arma::vec & *maxVals,* const arma::vec & *minVals,* const size_t *totalPoints,* const size_t *start,* const size_t *end*)

Create a child node of a density estimation tree given the bounding box specified by maxVals and minVals, using the size given in start and end, and calculating the error with the total number of points given.

Children of this node will not be created recursively.

Parameters

maxVals	Upper bound of bounding box.
minVals	Lower bound of bounding box.
start	Start of points represented by this node in the data matrix.
end	End of points represented by this node in the data matrix.

21.6.2.6 mlpack::det::DTree::~DTree ( )

Clean up memory allocated by the tree.

21.6.3 Member Function Documentation

21.6.3.1 double mlpack::det::DTree::AlphaUpper ( ) const [inline]

Return the upper part of the alpha sum.

Definition at line 284 of file dtree.hpp.

References alphaUpper.

21.6.3.2 double mlpack::det::DTree::ComputeValue (const arma::vec & *query*) const

Compute the logarithm of the density estimate of a given query point.

Parameters

query	Point to estimate density of.

21.6.3.3 void mlpack::det::DTree::ComputeVariableImportance (arma::vec & *importances*) const

Compute the variable importance of each dimension in the learned tree.

Parameters

importances	Vector to store the calculated importances in.

21.6.3.4 size_t mlpack::det::DTree::End ( ) const [inline]

Return the first index of a point not contained in this node.

Definition at line 261 of file dtree.hpp.

References end.

21.6.3.5 int mlpack::det::DTree::FindBucket (const arma::vec & *query*) const

Return the tag of the leaf containing the query.

This is useful for generating class memberships.

Parameters

query	Query to search for.

21.6.3.6 bool mlpack::det::DTree::FindSplit (const arma::mat & *data*, size_t & *splitDim*, double & *splitValue*, double & *leftError*, double & *rightError*, const size_t *maxLeafSize* = 10, const size_t *minLeafSize* = 5) const [private]

Find the dimension to split on.

21.6.3.7 double mlpack::det::DTree::Grow (arma::mat & *data*, arma::Col< size_t > & *oldFromNew*, const bool *useVolReg* = false, const size_t *maxLeafSize* = 10, const size_t *minLeafSize* = 5)

Greedily expand the tree.

The points in the dataset will be reordered during tree growth.

Parameters

data	Dataset to build tree on.
oldFromNew	Mappings from old points to new points.
useVolReg	If true, volume regularization is used.
maxLeafSize	Maximum size of a leaf.
minLeafSize	Minimum size of a leaf.

21.6.3.8 DTree* mlpack::det::DTree::Left ( ) const [inline]

Return the left child.

Definition at line 278 of file dtree.hpp.

References left.

21.6.3.9 double mlpack::det::DTree::LogNegativeError (const size_t *totalPoints*) const

Compute the log-negative-error for this point, given the total number of points in the dataset.

Parameters

totalPoints	Total number of points in the dataset.

21.6.3.10 double mlpack::det::DTree::LogNegError ( ) const [inline]

Return the log negative error of this node.

Definition at line 267 of file dtree.hpp.

References logNegError.

21.6.3.11 double mlpack::det::DTree::LogVolume ( ) const [inline]

Return the inverse of the volume of this node.

Definition at line 276 of file dtree.hpp.

References logVolume.

21.6.3.12 const arma::vec& mlpack::det::DTree::MaxVals ( ) const [inline]

Return the maximum values.

Definition at line 287 of file dtree.hpp.

References maxVals.

21.6.3.13 arma::vec& mlpack::det::DTree::MaxVals ( ) [inline]

Modify the maximum values.

Definition at line 289 of file dtree.hpp.

References maxVals.

21.6.3.14 const arma::vec& mlpack::det::DTree::MinVals ( ) const [inline]

Return the minimum values.

Definition at line 292 of file dtree.hpp.

References minVals.

21.6.3.15 arma::vec& mlpack::det::DTree::MinVals ( ) [inline]

Modify the minimum values.

Definition at line 294 of file dtree.hpp.

References minVals.

21.6.3.16 double mlpack::det::DTree::PruneAndUpdate (const double *oldAlpha,* const size_t *points,* const bool *useVolReg* = false)

Perform alpha pruning on a tree.

Returns the new value of alpha.

Parameters

oldAlpha	Old value of alpha.
points	Total number of points in dataset.
useVolReg	If true, volume regularization is used.

Returns

New value of alpha.

21.6.3.17 double mlpack::det::DTree::Ratio ( ) const [inline]

Return the ratio of points in this node to the points in the whole dataset.

Definition at line 274 of file dtree.hpp.

References ratio.

21.6.3.18 DTree* mlpack::det::DTree::Right ( ) const [inline]

Return the right child.

Definition at line 280 of file dtree.hpp.

References right.

21.6.3.19 bool mlpack::det::DTree::Root ( ) const [inline]

Return whether or not this is the root of the tree.

Definition at line 282 of file dtree.hpp.

References root.

21.6.3.20 size_t mlpack::det::DTree::SplitData (arma::mat & *data,* const size_t *splitDim,* const double *splitValue,* arma::Col< size_t > & *oldFromNew*) const [private]

Split the data, returning the number of points left of the split.

21.6.3.21 size_t mlpack::det::DTree::SplitDim ( ) const [inline]

Return the split dimension of this node.

Definition at line 263 of file dtree.hpp.

References splitDim.

21.6.3.22 double mlpack::det::DTree::SplitValue ( ) const [inline]

Return the split value of this node.

Definition at line 265 of file dtree.hpp.

References splitValue.

21.6.3.23 size_t mlpack::det::DTree::Start ( ) const [inline]

Return the starting index of points contained in this node.

Definition at line 259 of file dtree.hpp.

References start.

21.6.3.24 size_t mlpack::det::DTree::SubtreeLeaves ( ) const [inline]

Return the number of leaves which are descendants of this node.

Definition at line 271 of file dtree.hpp.

References subtreeLeaves.

21.6.3.25 double mlpack::det::DTree::SubtreeLeavesLogNegError ( ) const [inline]

Return the log negative error of all descendants of this node.

Definition at line 269 of file dtree.hpp.

References subtreeLeavesLogNegError.

21.6.3.26 int mlpack::det::DTree::TagTree (const int *tag* = 0)

Index the buckets for possible usage later; this results in every leaf in the tree having a specific tag (accessible with BucketTag()).

This function calls itself recursively.

Parameters

tag	Tag for the next leaf; leave at 0 for the initial call.

21.6.3.27 bool mlpack::det::DTree::WithinRange (const arma::vec & *query*) const

Return whether a query point is within the range of this node.

21.6.3.28 void mlpack::det::DTree::WriteTree (FILE * *fp*, const size_t *level* = 0) const

Print the tree in a depth-first manner (this function is called recursively).

Parameters

fp	File to write the tree to.
level	Level of the tree (should start at 0).

21.6.4 Member Data Documentation

21.6.4.1 double mlpack::det::DTree::alphaUpper [private]

Upper part of alpha sum; used for pruning.

Definition at line 250 of file dtree.hpp.

Referenced by AlphaUpper().

21.6.4.2 int mlpack::det::DTree::bucketTag [private]

The tag for the leaf, used for hashing points.

Definition at line 247 of file dtree.hpp.

21.6.4.3 size_t mlpack::det::DTree::end [private]

The index of the last point in the dataset contained in this node (and its children).

Definition at line 215 of file dtree.hpp.

Referenced by End().

21.6.4.4 DTree* mlpack::det::DTree::left [private]

The left child.

Definition at line 253 of file dtree.hpp.

Referenced by Left().

21.6.4.5 double mlpack::det::DTree::logNegError [private]

log-negative-L2-error of the node.

Definition at line 229 of file dtree.hpp.

Referenced by LogNegError().

21.6.4.6 double mlpack::det::DTree::logVolume [private]

The logarithm of the volume of the node.

Definition at line 244 of file dtree.hpp.

Referenced by LogVolume().

21.6.4.7 arma::vec mlpack::det::DTree::maxVals [private]

Upper half of bounding box for this node.

Definition at line 218 of file dtree.hpp.

Referenced by MaxVals().

21.6.4.8 arma::vec mlpack::det::DTree::minVals [private]

Lower half of bounding box for this node.

Definition at line 220 of file dtree.hpp.

Referenced by MinVals().

21.6.4.9 double mlpack::det::DTree::ratio [private]

Ratio of the number of points in the node to the total number of points.

Definition at line 241 of file dtree.hpp.

Referenced by Ratio().

21.6.4.10 DTree* mlpack::det::DTree::right [private]

The right child.

Definition at line 255 of file dtree.hpp.

Referenced by Right().

21.6.4.11 bool mlpack::det::DTree::root [private]

If true, this node is the root of the tree.

Definition at line 238 of file dtree.hpp.

Referenced by Root().

21.6.4.12 size_t mlpack::det::DTree::splitDim [private]

The splitting dimension for this node.

Definition at line 223 of file dtree.hpp.

Referenced by SplitDim().

21.6.4.13 double mlpack::det::DTree::splitValue [private]

The split value on the splitting dimension for this node.

Definition at line 226 of file dtree.hpp.

Referenced by SplitValue().

21.6.4.14 size_t mlpack::det::DTree::start [private]

The index of the first point in the dataset contained in this node (and its children).

Definition at line 212 of file dtree.hpp.

Referenced by Start().

21.6.4.15 size_t mlpack::det::DTree::subtreeLeaves [private]

Number of leaves of the subtree.

Definition at line 235 of file dtree.hpp.

Referenced by SubtreeLeaves().

21.6.4.16 double mlpack::det::DTree::subtreeLeavesLogNegError [private]

Sum of the error of the leaves of the subtree.

Definition at line 232 of file dtree.hpp.

Referenced by SubtreeLeavesLogNegError().

The documentation for this class was generated from the following file:

- src/mlpack/methods/det/**dtree.hpp**

21.7 mlpack::distribution::DiscreteDistribution Class Reference

A discrete distribution where the only observations are discrete observations.

Public Member Functions

- **DiscreteDistribution** ()

  Default constructor, which creates a distribution that has no observations.
- **DiscreteDistribution** (const size_t numObservations)

  Define the discrete distribution as having numObservations possible observations.
- **DiscreteDistribution** (const arma::vec &**probabilities**)

  Define the discrete distribution as having the given probabilities for each observation.
- size_t **Dimensionality** () const

  Get the dimensionality of the distribution.
- void **Estimate** (const arma::mat &observations)

  Estimate the probability distribution directly from the given observations.
- void **Estimate** (const arma::mat &observations, const arma::vec &**probabilities**)

  Estimate the probability distribution from the given observations, taking into account the probability of each observation actually being from this distribution.
- const arma::vec & **Probabilities** () const

  Return the vector of probabilities.
- arma::vec & **Probabilities** ()

  Modify the vector of probabilities.

- double **Probability** (const arma::vec &observation) const

  Return the probability of the given observation.

- arma::vec **Random** () const

  Return a randomly generated observation (one-dimensional vector; one observation) according to the probability distribution defined by this object.

- std::string **ToString** () const

Private Attributes

- arma::vec **probabilities**

21.7.1 Detailed Description

A discrete distribution where the only observations are discrete observations.

This is useful (for example) with discrete Hidden Markov Models, where observations are non-negative integers representing specific emissions.

No bounds checking is performed for observations, so if an invalid observation is passed (i.e. observation $>$ numObservations), a crash will probably occur.

This distribution only supports one-dimensional observations, so when passing an arma::vec as an observation, it should only have one dimension (vec.n_rows == 1). Any additional dimensions will simply be ignored.

Note

This class, like every other class in MLPACK, uses arma::vec to represent observations. While a discrete distribution only has positive integers (size_t) as observations, these can be converted to doubles (which is what arma::vec holds). This distribution internally converts those doubles back into size_t before comparisons.

Definition at line 53 of file discrete_distribution.hpp.

21.7.2 Constructor & Destructor Documentation

21.7.2.1 mlpack::distribution::DiscreteDistribution::DiscreteDistribution ( ) [inline]

Default constructor, which creates a distribution that has no observations.

Definition at line 59 of file discrete_distribution.hpp.

21.7.2.2 mlpack::distribution::DiscreteDistribution::DiscreteDistribution (const size_t *numObservations*) [inline]

Define the discrete distribution as having numObservations possible observations.

The probability in each state will be set to (1 / numObservations).

Parameters

numObservations	Number of possible observations this distribution can have.

Definition at line 69 of file discrete_distribution.hpp.

21.7.2.3 mlpack::distribution::DiscreteDistribution::DiscreteDistribution (const arma::vec & *probabilities*) [inline]

Define the discrete distribution as having the given probabilities for each observation.

Parameters

probabilities	Probabilities of each possible observation.

Definition at line 79 of file discrete_distribution.hpp.

21.7.3 Member Function Documentation

21.7.3.1 size_t mlpack::distribution::DiscreteDistribution::Dimensionality ( ) const [inline]

Get the dimensionality of the distribution.

Definition at line 95 of file discrete_distribution.hpp.

21.7.3.2 void mlpack::distribution::DiscreteDistribution::Estimate (const arma::mat & *observations*)

Estimate the probability distribution directly from the given observations.

If any of the observations is greater than numObservations, a crash is likely to occur.

Parameters

observations	List of observations.

21.7.3.3 void mlpack::distribution::DiscreteDistribution::Estimate (const arma::mat & *observations,* const arma::vec & *probabilities*)

Estimate the probability distribution from the given observations, taking into account the probability of each observation actually being from this distribution.

Parameters

observations	List of observations.
probabilities	List of probabilities that each observation is actually from this distribution.

21.7.3.4 const arma::vec& mlpack::distribution::DiscreteDistribution::Probabilities ( ) const [inline]

Return the vector of probabilities.

Definition at line 153 of file discrete_distribution.hpp.

References probabilities.

21.7.3.5 arma::vec& mlpack::distribution::DiscreteDistribution::Probabilities ( ) [inline]

Modify the vector of probabilities.

Definition at line 155 of file discrete_distribution.hpp.

References probabilities.

21.7.3.6 double mlpack::distribution::DiscreteDistribution::Probability (const arma::vec & *observation*) const [inline]

Return the probability of the given observation.

If the observation is greater than the number of possible observations, then a crash will probably occur – bounds checking is not performed.

Parameters

observation	Observation to return the probability of.

Returns

Probability of the given observation.

Definition at line 105 of file discrete_distribution.hpp.

References mlpack::Log::Debug, and probabilities.

21.7.3.7 arma::vec mlpack::distribution::DiscreteDistribution::Random ( ) const

Return a randomly generated observation (one-dimensional vector; one observation) according to the probability distribution defined by this object.

Returns

Random observation.

21.7.3.8 std::string mlpack::distribution::DiscreteDistribution::ToString ( ) const

21.7.4 Member Data Documentation

21.7.4.1 arma::vec mlpack::distribution::DiscreteDistribution::probabilities [private]

Definition at line 163 of file discrete_distribution.hpp.

Referenced by Probabilities(), and Probability().

The documentation for this class was generated from the following file:

- src/mlpack/core/dists/**discrete_distribution.hpp**

21.8 mlpack::distribution::GaussianDistribution Class Reference

A single multivariate Gaussian distribution.

Public Member Functions

- **GaussianDistribution** ()

  Default constructor, which creates a Gaussian with zero dimension.
- **GaussianDistribution** (const size_t dimension)

  Create a Gaussian distribution with zero mean and identity covariance with the given dimensionality.
- **GaussianDistribution** (const arma::vec &**mean**, const arma::mat &**covariance**)

  Create a Gaussian distribution with the given mean and covariance.
- const arma::mat & **Covariance** () const

Return the covariance matrix.

- arma::mat & **Covariance** ()

  Return a modifiable copy of the covariance.

- size_t **Dimensionality** () const

  Return the dimensionality of this distribution.

- void **Estimate** (const arma::mat &observations)

  Estimate the Gaussian distribution directly from the given observations.

- void **Estimate** (const arma::mat &observations, const arma::vec &probabilities)

  Estimate the Gaussian distribution from the given observations, taking into account the probability of each observation actually being from this distribution.

- const arma::vec & **Mean** () const

  Return the mean.

- arma::vec & **Mean** ()

  Return a modifiable copy of the mean.

- double **Probability** (const arma::vec &observation) const

  Return the probability of the given observation.

- arma::vec **Random** () const

  Return a randomly generated observation according to the probability distribution defined by this object.

- std::string **ToString** () const

  Returns a string representation of this object.

Private Attributes

- arma::mat **covariance**

  Covariance of the distribution.

- arma::vec **mean**

  Mean of the distribution.

21.8.1 Detailed Description

A single multivariate Gaussian distribution.

Definition at line 35 of file gaussian_distribution.hpp.

21.8.2 Constructor & Destructor Documentation

21.8.2.1 mlpack::distribution::GaussianDistribution::GaussianDistribution ( ) [inline]

Default constructor, which creates a Gaussian with zero dimension.

Definition at line 47 of file gaussian_distribution.hpp.

21.8.2.2 mlpack::distribution::GaussianDistribution::GaussianDistribution (const size_t *dimension*) [inline]

Create a Gaussian distribution with zero mean and identity covariance with the given dimensionality.

Definition at line 53 of file gaussian_distribution.hpp.

21.8.2.3 mlpack::distribution::GaussianDistribution::GaussianDistribution (const arma::vec & *mean,* const arma::mat & *covariance*) [inline]

Create a Gaussian distribution with the given mean and covariance.

Definition at line 61 of file gaussian_distribution.hpp.

21.8.3 Member Function Documentation

21.8.3.1 const arma::mat& mlpack::distribution::GaussianDistribution::Covariance ( ) const [inline]

Return the covariance matrix.

Definition at line 111 of file gaussian_distribution.hpp.

References covariance.

21.8.3.2 arma::mat& mlpack::distribution::GaussianDistribution::Covariance ( ) [inline]

Return a modifiable copy of the covariance.

Definition at line 116 of file gaussian_distribution.hpp.

References covariance.

21.8.3.3 size_t mlpack::distribution::GaussianDistribution::Dimensionality ( ) const [inline]

Return the dimensionality of this distribution.

Definition at line 65 of file gaussian_distribution.hpp.

References mean.

21.8.3.4 void mlpack::distribution::GaussianDistribution::Estimate (const arma::mat & *observations*)

Estimate the Gaussian distribution directly from the given observations.

Parameters

observations	List of observations.

21.8.3.5 void mlpack::distribution::GaussianDistribution::Estimate (const arma::mat & *observations,* const arma::vec & *probabilities*)

Estimate the Gaussian distribution from the given observations, taking into account the probability of each observation actually being from this distribution.

21.8.3.6 const arma::vec& mlpack::distribution::GaussianDistribution::Mean ( ) const [inline]

Return the mean.

Definition at line 101 of file gaussian_distribution.hpp.

References mean.

21.8.3.7 arma::vec& mlpack::distribution::GaussianDistribution::Mean ( ) [inline]

Return a modifiable copy of the mean.

Definition at line 106 of file gaussian_distribution.hpp.

References mean.

21.8.3.8 double mlpack::distribution::GaussianDistribution::Probability (const arma::vec & *observation*) const [inline]

Return the probability of the given observation.

Definition at line 70 of file gaussian_distribution.hpp.

References covariance, mean, and mlpack::gmm::phi().

21.8.3.9 arma::vec mlpack::distribution::GaussianDistribution::Random ( ) const

Return a randomly generated observation according to the probability distribution defined by this object.

Returns

Random observation from this Gaussian distribution.

21.8.3.10 std::string mlpack::distribution::GaussianDistribution::ToString ( ) const

Returns a string representation of this object.

21.8.4 Member Data Documentation

21.8.4.1 arma::mat mlpack::distribution::GaussianDistribution::covariance [private]

Covariance of the distribution.

Definition at line 41 of file gaussian_distribution.hpp.

Referenced by Covariance(), and Probability().

21.8.4.2 arma::vec mlpack::distribution::GaussianDistribution::mean [private]

Mean of the distribution.

Definition at line 39 of file gaussian_distribution.hpp.

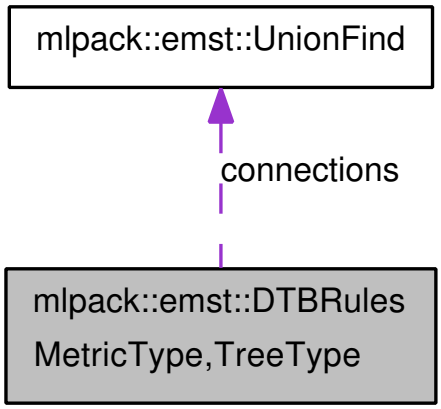
Referenced by Dimensionality(), Mean(), and Probability().

The documentation for this class was generated from the following file:

- src/mlpack/core/dists/**gaussian_distribution.hpp**

21.9 mlpack::emst::DTBRules< MetricType, TreeType > Class Template Reference

Collaboration diagram for mlpack::emst::DTBRules< MetricType, TreeType >:



Public Member Functions

- **DTBRules** (const arma::mat &**dataSet**, **UnionFind** &**connections**, arma::vec &**neighborsDistances**, arma::Col< size_t > &**neighborsInComponent**, arma::Col< size_t > &**neighborsOutComponent**, MetricType &**metric**)
- double **BaseCase** (const size_t queryIndex, const size_t referenceIndex)
- double **Rescore** (const size_t queryIndex, TreeType &referenceNode, const double oldScore)

  Re-evaluate the score for recursion order.
- double **Rescore** (TreeType &queryNode, TreeType &referenceNode, const double oldScore) const

  Re-evaluate the score for recursion order.
- double **Score** (const size_t queryIndex, TreeType &referenceNode)

  Get the score for recursion order.
- double **Score** (const size_t queryIndex, TreeType &referenceNode, const double baseCaseResult)

  Get the score for recursion order, passing the base case result (in the situation where it may be needed to calculate the recursion order).
- double **Score** (TreeType &queryNode, TreeType &referenceNode) const

  Get the score for recursion order.
- double **Score** (TreeType &queryNode, TreeType &referenceNode, const double baseCaseResult) const

  Get the score for recursion order, passing the base case result (in the situation where it may be needed to calculate the recursion order).

Private Member Functions

- double **CalculateBound** (TreeType &queryNode) const

  Update the bound for the given query node.

Private Attributes

- **UnionFind** & **connections**

  Stores the tree structure so far.

- const arma::mat & **dataSet**

  The data points.
- MetricType & **metric**

  The instantiated metric.
- arma::vec & **neighborsDistances**

  The distance to the candidate nearest neighbor for each component.
- arma::Col< size_t > & **neighborsInComponent**

  The index of the point in the component that is an endpoint of the candidate edge.
- arma::Col< size_t > & **neighborsOutComponent**

  The index of the point outside of the component that is an endpoint of the candidate edge.

21.9.1 Detailed Description

template<typename MetricType, typename TreeType>class mlpack::emst::DTBRules< MetricType, TreeType >

Definition at line 31 of file dtb_rules.hpp.

21.9.2 Constructor & Destructor Documentation

21.9.2.1 template<typename MetricType , typename TreeType > mlpack::emst::DTBRules< MetricType, TreeType >::DTBRules (const arma::mat & *dataSet*, UnionFind & *connections*, arma::vec & *neighborsDistances*, arma::Col< size_t > & *neighborsInComponent*, arma::Col< size_t > & *neighborsOutComponent*, MetricType & *metric*)

21.9.3 Member Function Documentation

21.9.3.1 template<typename MetricType , typename TreeType > double mlpack::emst::DTBRules< MetricType, TreeType >::BaseCase (const size_t *queryIndex*, const size_t *referenceIndex*)

21.9.3.2 template<typename MetricType , typename TreeType > double mlpack::emst::DTBRules< MetricType, TreeType >::CalculateBound (TreeType & *queryNode*) const [inline], [private]

Update the bound for the given query node.

21.9.3.3 template<typename MetricType , typename TreeType > double mlpack::emst::DTBRules< MetricType, TreeType >::Rescore (const size_t *queryIndex*, TreeType & *referenceNode*, const double *oldScore*)

Re-evaluate the score for recursion order.

A low score indicates priority for recursion, while DBL_MAX indicates that the node should not be recursed into at all (it should be pruned). This is used when the score has already been calculated, but another recursion may have modified the bounds for pruning. So the old score is checked against the new pruning bound.

Parameters

queryIndex	Index of query point.
referenceNode	Candidate node to be recursed into.

oldScore	Old score produced by **Score()** (p. 167) (or **Rescore()** (p. 166)).

21.9.3.4 template<typename MetricType , typename TreeType > double mlpack::emst::DTBRules< MetricType, TreeType >::Rescore (TreeType & *queryNode,* TreeType & *referenceNode,* const double *oldScore*) const

Re-evaluate the score for recursion order.

A low score indicates priority for recursion, while DBL_MAX indicates that the node should not be recursed into at all (it should be pruned). This is used when the score has already been calculated, but another recursion may have modified the bounds for pruning. So the old score is checked against the new pruning bound.

Parameters

queryNode	Candidate query node to recurse into.
referenceNode	Candidate reference node to recurse into.
oldScore	Old score produced by Socre() (or **Rescore()** (p. 166)).

21.9.3.5 template<typename MetricType , typename TreeType > double mlpack::emst::DTBRules< MetricType, TreeType >::Score (const size_t *queryIndex,* TreeType & *referenceNode*)

Get the score for recursion order.

A low score indicates priority for recursion, while DBL_MAX indicates that the node should not be recursed into at all (it should be pruned).

Parameters

queryIndex	Index of query point.
referenceNode	Candidate node to be recursed into.

21.9.3.6 template<typename MetricType , typename TreeType > double mlpack::emst::DTBRules< MetricType, TreeType >::Score (const size_t *queryIndex,* TreeType & *referenceNode,* const double *baseCaseResult*)

Get the score for recursion order, passing the base case result (in the situation where it may be needed to calculate the recursion order).

A low score indicates priority for recursion, while DBL_MAX indicates that the node should not be recursed into at all (it should be pruned).

Parameters

queryIndex	Index of query point.
referenceNode	Candidate node to be recursed into.
baseCaseResult	Result of BaseCase(queryIndex, referenceNode).

21.9.3.7 template<typename MetricType , typename TreeType > double mlpack::emst::DTBRules< MetricType, TreeType >::Score (TreeType & *queryNode,* TreeType & *referenceNode*) const

Get the score for recursion order.

A low score indicates priority for recursionm while DBL_MAX indicates that the node should not be recursed into at all (it should be pruned).

Parameters

queryNode	Candidate query node to recurse into.
referenceNode	Candidate reference node to recurse into.

21.9.3.8 template<typename MetricType , typename TreeType > double mlpack::emst::DTBRules< MetricType, TreeType >::Score (TreeType & *queryNode,* TreeType & *referenceNode,* const double *baseCaseResult*) const

Get the score for recursion order, passing the base case result (in the situation where it may be needed to calculate the recursion order).

A low score indicates priority for recursion, while DBL_MAX indicates that the node should not be recursed into at all (it should be pruned).

Parameters

queryNode	Candidate query node to recurse into.
referenceNode	Candidate reference node to recurse into.
baseCaseResult	Result of BaseCase(queryIndex, referenceNode).

21.9.4 Member Data Documentation

21.9.4.1 template<typename MetricType , typename TreeType > UnionFind& mlpack::emst::DTBRules< MetricType, TreeType >::connections [private]

Stores the tree structure so far.

Definition at line 126 of file dtb_rules.hpp.

21.9.4.2 template<typename MetricType , typename TreeType > const arma::mat& mlpack::emst::DTBRules< MetricType, TreeType >::dataSet [private]

The data points.

Definition at line 123 of file dtb_rules.hpp.

21.9.4.3 template<typename MetricType , typename TreeType > MetricType& mlpack::emst::DTBRules< MetricType, TreeType >::metric [private]

The instantiated metric.

Definition at line 140 of file dtb_rules.hpp.

21.9.4.4 template<typename MetricType , typename TreeType > arma::vec& mlpack::emst::DTBRules< MetricType, TreeType >::neighborsDistances [private]

The distance to the candidate nearest neighbor for each component.

Definition at line 129 of file dtb_rules.hpp.

21.9.4.5 template<typename MetricType , typename TreeType > arma::Col<size_t>& mlpack::emst::DTBRules< MetricType, TreeType >::neighborsInComponent [private]

The index of the point in the component that is an endpoint of the candidate edge.

Definition at line 133 of file dtb_rules.hpp.

21.9.4.6 template<typename MetricType , typename TreeType > arma::Col<size_t>& mlpack::emst::DTBRules< MetricType, TreeType >::neighborsOutComponent [private]

The index of the point outside of the component that is an endpoint of the candidate edge.

Definition at line 137 of file dtb_rules.hpp.

The documentation for this class was generated from the following file:

- src/mlpack/methods/emst/**dtb_rules.hpp**

21.10 mlpack::emst::DTBStat Class Reference

A statistic for use with MLPACK trees, which stores the upper bound on distance to nearest neighbors and the component which this node belongs to.

Public Member Functions

- **DTBStat** ()

  A generic initializer.
- template<typename TreeType >
  DTBStat (const TreeType &node)

  This is called when a node is finished initializing.
- double **Bound** () const

  Get the total bound for pruning.
- double & **Bound** ()

  Modify the total bound for pruning.
- int **ComponentMembership** () const

  Get the component membership of this node.
- int & **ComponentMembership** ()

  Modify the component membership of this node.
- double **MaxNeighborDistance** () const

  Get the maximum neighbor distance.
- double & **MaxNeighborDistance** ()

  Modify the maximum neighbor distance.
- double **MinNeighborDistance** () const

  Get the minimum neighbor distance.
- double & **MinNeighborDistance** ()

  Modify the minimum neighbor distance.

Private Attributes

- double **bound**

  Total bound for pruning.
- int **componentMembership**

  The index of the component that all points in this node belong to.
- double **maxNeighborDistance**

  Upper bound on the distance to the nearest neighbor of any point in this node.
- double **minNeighborDistance**

  Lower bound on the distance to the nearest neighbor of any point in this node.

21.10.1 Detailed Description

A statistic for use with MLPACK trees, which stores the upper bound on distance to nearest neighbors and the component which this node belongs to.

Definition at line 34 of file dtb_stat.hpp.

21.10.2 Constructor & Destructor Documentation

21.10.2.1 mlpack::emst::DTBStat::DTBStat ( )

A generic initializer.

Sets the maximum neighbor distance to its default, and the component membership to -1 (no component).

21.10.2.2 template<typename TreeType > mlpack::emst::DTBStat::DTBStat (const TreeType & *node*)

This is called when a node is finished initializing.

We set the maximum neighbor distance to its default, and if possible, we set the component membership of the node (if it has only one point and no children).

Parameters

node	Node that has been finished.

21.10.3 Member Function Documentation

21.10.3.1 double mlpack::emst::DTBStat::Bound ( ) const [inline]

Get the total bound for pruning.

Definition at line 82 of file dtb_stat.hpp.

References bound.

21.10.3.2 double& mlpack::emst::DTBStat::Bound ( ) [inline]

Modify the total bound for pruning.

Definition at line 84 of file dtb_stat.hpp.

References bound.

21.10.3.3 int mlpack::emst::DTBStat::ComponentMembership ( ) const [inline]

Get the component membership of this node.

Definition at line 87 of file dtb_stat.hpp.

References componentMembership.

21.10.3.4 int& mlpack::emst::DTBStat::ComponentMembership ( ) [inline]

Modify the component membership of this node.

Definition at line 89 of file dtb_stat.hpp.

References componentMembership.

21.10.3.5 double mlpack::emst::DTBStat::MaxNeighborDistance ( ) const [inline]

Get the maximum neighbor distance.

Definition at line 72 of file dtb_stat.hpp.

References maxNeighborDistance.

21.10.3.6 double& mlpack::emst::DTBStat::MaxNeighborDistance ( ) [inline]

Modify the maximum neighbor distance.

Definition at line 74 of file dtb_stat.hpp.

References maxNeighborDistance.

21.10.3.7 double mlpack::emst::DTBStat::MinNeighborDistance ( ) const [inline]

Get the minimum neighbor distance.

Definition at line 77 of file dtb_stat.hpp.

References minNeighborDistance.

21.10.3.8 double& mlpack::emst::DTBStat::MinNeighborDistance ( ) [inline]

Modify the minimum neighbor distance.

Definition at line 79 of file dtb_stat.hpp.

References minNeighborDistance.

21.10.4 Member Data Documentation

21.10.4.1 double mlpack::emst::DTBStat::bound [private]

Total bound for pruning.

Definition at line 46 of file dtb_stat.hpp.

Referenced by Bound().

21.10.4.2 int mlpack::emst::DTBStat::componentMembership [private]

The index of the component that all points in this node belong to.

This is the same index returned by **UnionFind** (p. 183) for all points in this node. If points in this node are in different components, this value will be negative.

Definition at line 52 of file dtb_stat.hpp.

Referenced by ComponentMembership().

21.10.4.3 double mlpack::emst::DTBStat::maxNeighborDistance [private]

Upper bound on the distance to the nearest neighbor of any point in this node.

Definition at line 39 of file dtb_stat.hpp.

Referenced by MaxNeighborDistance().

21.10.4.4 double mlpack::emst::DTBStat::minNeighborDistance [private]

Lower bound on the distance to the nearest neighbor of any point in this node.

Definition at line 43 of file dtb_stat.hpp.

Referenced by MinNeighborDistance().

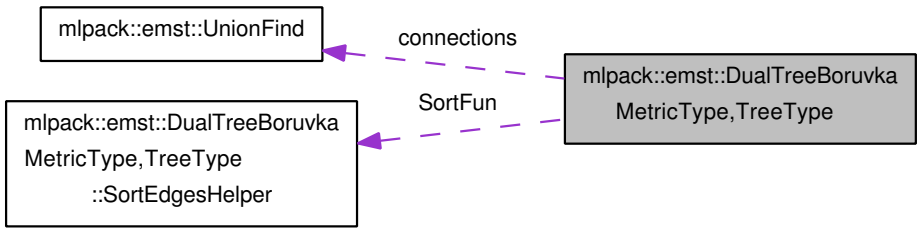
The documentation for this class was generated from the following file:

- src/mlpack/methods/emst/**dtb_stat.hpp**

21.11 mlpack::emst::DualTreeBoruvka< MetricType, TreeType > Class Template Reference

Performs the MST calculation using the Dual-Tree Boruvka algorithm, using any type of tree.

Collaboration diagram for mlpack::emst::DualTreeBoruvka< MetricType, TreeType >:



Classes

- struct **SortEdgesHelper**

  For sorting the edge list after the computation.

Public Member Functions

- **DualTreeBoruvka** (const typename TreeType::Mat &dataset, const bool **naive**=false, const size_t leafSize=1, const MetricType **metric**=MetricType())

  Create the tree from the given dataset.
- **DualTreeBoruvka** (TreeType *__**tree**, const typename TreeType::Mat &dataset, const MetricType **metric**=MetricType())

  *Create the **DualTreeBoruvka** (p. 172) object with an already initialized tree.*
- ~**DualTreeBoruvka** ()

  Delete the tree, if it was created inside the object.
- void **ComputeMST** (arma::mat &results)

  Iteratively find the nearest neighbor of each component until the MST is complete.

Private Member Functions

- void **AddAllEdges** ()

  Adds all the edges found in one iteration to the list of neighbors.
- void **AddEdge** (const size_t e1, const size_t e2, const double distance)

  Adds a single edge to the edge list.
- void **Cleanup** ()

  The values stored in the tree must be reset on each iteration.
- void **CleanupHelper** (TreeType *__**tree**)

  This function resets the values in the nodes of the tree nearest neighbor distance, and checks for fully connected nodes.
- void **EmitResults** (arma::mat &results)

  Unpermute the edge list and output it to results.

Private Attributes

- **UnionFind connections**

  Connections.
- TreeType::Mat & **data**

  Reference to the data (this is what should be used for accessing data).
- TreeType::Mat **dataCopy**

  Copy of the data (if necessary).
- std::vector< **EdgePair** > **edges**

  Edges.
- MetricType **metric**

  The instantiated metric.
- bool **naive**

  Indicates whether or not O(n^2) naive mode will be used.
- arma::vec **neighborsDistances**

  List of edge distances.
- arma::Col< size_t > **neighborsInComponent**

  List of edge nodes.
- arma::Col< size_t > **neighborsOutComponent**

  List of edge nodes.
- std::vector< size_t > **oldFromNew**

Permutations of points during tree building.

- bool **ownTree**

  Indicates whether or not we "own" the tree.

- struct **mlpack::emst::DualTreeBoruvka::SortEdgesHelper SortFun**
- double **totalDist**

  Total distance of the tree.

- TreeType * **tree**

  Pointer to the root of the tree.

21.11.1 Detailed Description

template<typename MetricType = metric::EuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, DTBStat>>class mlpack::emst::DualTreeBoruvka< MetricType, TreeType >

Performs the MST calculation using the Dual-Tree Boruvka algorithm, using any type of tree.

For more information on the algorithm, see the following citation:

```
@inproceedings{
  author = {March, W.B., Ram, P., and Gray, A.G.},
  title = {{Fast Euclidean Minimum Spanning Tree: Algorithm, Analysis,
     Applications.}},
  booktitle = {Proceedings of the 16th ACM SIGKDD International Conference
     on Knowledge Discovery and Data Mining}
  series = {KDD 2010},
  year = {2010}
}
```

General usage of this class might be like this:

```
extern arma::mat data; // We want to find the MST of this dataset.
DualTreeBoruvka<> dtb(data); // Create the tree with default options.

// Find the MST.
arma::mat mstResults;
dtb.ComputeMST(mstResults);
```

More advanced usage of the class can use different types of trees, pass in an already-built tree, or compute the MST using the O(n^2) naive algorithm.

Template Parameters

MetricType	The metric to use. IMPORTANT: this hasn't really been tested with anything other than the L2 metric, so user beware. Note that the tree type needs to compute bounds using the same metric as the type specified here.
TreeType	Type of tree to use. Should use **DTBStat** (p. 169) as a statistic.

Definition at line 91 of file dtb.hpp.

21.11.2 Constructor & Destructor Documentation

21.11.2.1 template<typename MetricType = metric::EuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, DTBStat>> mlpack::emst::DualTreeBoruvka< MetricType, TreeType >::DualTreeBoruvka (const typename TreeType::Mat & *dataset*, const bool *naive* = `false`, const size_t *leafSize* = `1`, const MetricType *metric* = `MetricType()`)

Create the tree from the given dataset.

This copies the dataset to an internal copy, because tree-building modifies the dataset.

Parameters

data	Dataset to build a tree for.
naive	Whether the computation should be done in O(n^2) naive mode.
leafSize	The leaf size to be used during tree construction.

21.11.2.2 template<typename MetricType = metric::EuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, DTBStat>> mlpack::emst::DualTreeBoruvka< MetricType, TreeType >::DualTreeBoruvka (TreeType * *tree,* const typename TreeType::Mat & *dataset,* const MetricType *metric* = `MetricType()`)

Create the **DualTreeBoruvka** (p. 172) object with an already initialized tree.

This will not copy the dataset, and can save a little processing power. Naive mode is not available as an option for this constructor; instead, to run naive computation, construct a tree with all the points in one leaf (i.e. leafSize = number of points).

Note

Because tree-building (at least with BinarySpaceTree) modifies the ordering of a matrix, be sure you pass the modified matrix to this object! In addition, mapping the points of the matrix back to their original indices is not done when this constructor is used.

Parameters

tree	Pre-built tree.
dataset	Dataset corresponding to the pre-built tree.

21.11.2.3 template<typename MetricType = metric::EuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, DTBStat>> mlpack::emst::DualTreeBoruvka< MetricType, TreeType >::~DualTreeBoruvka ( )

Delete the tree, if it was created inside the object.

21.11.3 Member Function Documentation

21.11.3.1 template<typename MetricType = metric::EuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, DTBStat>> void mlpack::emst::DualTreeBoruvka< MetricType, TreeType >::AddAllEdges ( ) `[private]`

Adds all the edges found in one iteration to the list of neighbors.

21.11.3.2 template<typename MetricType = metric::EuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, DTBStat>> void mlpack::emst::DualTreeBoruvka< MetricType, TreeType >::AddEdge (const size_t *e1,* const size_t *e2,* const double *distance*) `[private]`

Adds a single edge to the edge list.

21.11.3.3 template<typename MetricType = metric::EuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, DTBStat>> void mlpack::emst::DualTreeBoruvka< MetricType, TreeType >::Cleanup ( ) [private]

The values stored in the tree must be reset on each iteration.

21.11.3.4 template<typename MetricType = metric::EuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, DTBStat>> void mlpack::emst::DualTreeBoruvka< MetricType, TreeType >::CleanupHelper (TreeType * *tree*) [private]

This function resets the values in the nodes of the tree nearest neighbor distance, and checks for fully connected nodes.

21.11.3.5 template<typename MetricType = metric::EuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, DTBStat>> void mlpack::emst::DualTreeBoruvka< MetricType, TreeType >::ComputeMST (arma::mat & *results*)

Iteratively find the nearest neighbor of each component until the MST is complete.

The results will be a 3xN matrix (with N equal to the number of edges in the minimum spanning tree). The first row will contain the lesser index of the edge; the second row will contain the greater index of the edge; and the third row will contain the distance between the two edges.

Parameters

results	Matrix which results will be stored in.

21.11.3.6 template<typename MetricType = metric::EuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, DTBStat>> void mlpack::emst::DualTreeBoruvka< MetricType, TreeType >::EmitResults (arma::mat & *results*) [private]

Unpermute the edge list and output it to results.

21.11.4 Member Data Documentation

21.11.4.1 template<typename MetricType = metric::EuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, DTBStat>> UnionFind mlpack::emst::DualTreeBoruvka< MetricType, TreeType >::connections [private]

Connections.

Definition at line 111 of file dtb.hpp.

21.11.4.2 template<typename MetricType = metric::EuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, DTBStat>> TreeType::Mat& mlpack::emst::DualTreeBoruvka< MetricType, TreeType >::data [private]

Reference to the data (this is what should be used for accessing data).

Definition at line 97 of file dtb.hpp.

21.11.4.3 template<typename MetricType = metric::EuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, DTBStat>> TreeType::Mat mlpack::emst::DualTreeBoruvka< MetricType, TreeType >::dataCopy [private]

Copy of the data (if necessary).

Definition at line 95 of file dtb.hpp.

21.11.4.4 template<typename MetricType = metric::EuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, DTBStat>> std::vector<EdgePair> mlpack::emst::DualTreeBoruvka< MetricType, TreeType >::edges [private]

Edges.

Definition at line 108 of file dtb.hpp.

21.11.4.5 template<typename MetricType = metric::EuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, DTBStat>> MetricType mlpack::emst::DualTreeBoruvka< MetricType, TreeType >::metric [private]

The instantiated metric.

Definition at line 126 of file dtb.hpp.

21.11.4.6 template<typename MetricType = metric::EuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, DTBStat>> bool mlpack::emst::DualTreeBoruvka< MetricType, TreeType >::naive [private]

Indicates whether or not O(n^2) naive mode will be used.

Definition at line 105 of file dtb.hpp.

21.11.4.7 template<typename MetricType = metric::EuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, DTBStat>> arma::vec mlpack::emst::DualTreeBoruvka< MetricType, TreeType >::neighborsDistances [private]

List of edge distances.

Definition at line 120 of file dtb.hpp.

21.11.4.8 template<typename MetricType = metric::EuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, DTBStat>> arma::Col<size_t> mlpack::emst::DualTreeBoruvka< MetricType, TreeType >::neighborsInComponent [private]

List of edge nodes.

Definition at line 116 of file dtb.hpp.

21.11.4.9 template<typename MetricType = metric::EuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, DTBStat>> arma::Col<size_t> mlpack::emst::DualTreeBoruvka< MetricType, TreeType >::neighborsOutComponent [private]

List of edge nodes.

Definition at line 118 of file dtb.hpp.

21.11.4.10 template<typename MetricType = metric::EuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, DTBStat>> std::vector<size_t> mlpack::emst::DualTreeBoruvka< MetricType, TreeType >::oldFromNew [private]

Permutations of points during tree building.

Definition at line 114 of file dtb.hpp.

21.11.4.11 template<typename MetricType = metric::EuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, DTBStat>> bool mlpack::emst::DualTreeBoruvka< MetricType, TreeType >::ownTree [private]

Indicates whether or not we "own" the tree.

Definition at line 102 of file dtb.hpp.

21.11.4.12 template<typename MetricType = metric::EuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, DTBStat>> struct mlpack::emst::DualTreeBoruvka::SortEdgesHelper mlpack::emst::DualTreeBoruvka< MetricType, TreeType >::SortFun [private]

21.11.4.13 template<typename MetricType = metric::EuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, DTBStat>> double mlpack::emst::DualTreeBoruvka< MetricType, TreeType >::totalDist [private]

Total distance of the tree.

Definition at line 123 of file dtb.hpp.

21.11.4.14 template<typename MetricType = metric::EuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, DTBStat>> TreeType* mlpack::emst::DualTreeBoruvka< MetricType, TreeType >::tree [private]

Pointer to the root of the tree.

Definition at line 100 of file dtb.hpp.

The documentation for this class was generated from the following file:

- src/mlpack/methods/emst/**dtb.hpp**

21.12 mlpack::emst::DualTreeBoruvka< MetricType, TreeType >::SortEdgesHelper Struct Reference

For sorting the edge list after the computation.

Public Member Functions

- bool **operator()** (const **EdgePair** &pairA, const **EdgePair** &pairB)

21.12.1 Detailed Description

template<typename MetricType = metric::EuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, DTBStat>>struct mlpack::emst::DualTreeBoruvka< MetricType, TreeType >::SortEdgesHelper

For sorting the edge list after the computation.

Definition at line 129 of file dtb.hpp.

21.12.2 Member Function Documentation

21.12.2.1 template<typename MetricType = metric::EuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, DTBStat>> bool mlpack::emst::DualTreeBoruvka< MetricType, TreeType >::SortEdgesHelper::operator() (const EdgePair & *pairA,* const EdgePair & *pairB*) [inline]

Definition at line 131 of file dtb.hpp.

References mlpack::emst::EdgePair::Distance().

The documentation for this struct was generated from the following file:

- src/mlpack/methods/emst/**dtb.hpp**

21.13 mlpack::emst::EdgePair Class Reference

An edge pair is simply two indices and a distance.

Public Member Functions

- **EdgePair** (const size_t **lesser**, const size_t **greater**, const double dist)
  *Initialize an **EdgePair** (*p. *180) with two indices and a distance.*
- double **Distance** () const
  Get the distance.
- double & **Distance** ()
  Modify the distance.
- size_t **Greater** () const
  Get the greater index.
- size_t & **Greater** ()
  Modify the greater index.
- size_t **Lesser** () const
  Get the lesser index.
- size_t & **Lesser** ()
  Modify the lesser index.

Private Attributes

- double **distance**
  Distance between two indices.
- size_t **greater**

Greater index.

- size_t **lesser**

  Lesser index.

21.13.1 Detailed Description

An edge pair is simply two indices and a distance.

It is used as the basic element of an edge list when computing a minimum spanning tree.

Definition at line 38 of file edge_pair.hpp.

21.13.2 Constructor & Destructor Documentation

21.13.2.1 mlpack::emst::EdgePair::EdgePair (const size_t *lesser,* const size_t *greater,* const double *dist*) [inline]

Initialize an **EdgePair** (p. 180) with two indices and a distance.

The indices are called lesser and greater, implying that they be sorted before calling Init. However, this is not necessary for functionality; it is just a way to keep the edge list organized in other code.

Definition at line 55 of file edge_pair.hpp.

References mlpack::Log::Assert().

21.13.3 Member Function Documentation

21.13.3.1 double mlpack::emst::EdgePair::Distance ( ) const [inline]

Get the distance.

Definition at line 73 of file edge_pair.hpp.

References distance.

Referenced by mlpack::emst::DualTreeBoruvka< MetricType, TreeType >::SortEdgesHelper::operator()().

21.13.3.2 double& mlpack::emst::EdgePair::Distance ( ) [inline]

Modify the distance.

Definition at line 75 of file edge_pair.hpp.

References distance.

21.13.3.3 size_t mlpack::emst::EdgePair::Greater ( ) const [inline]

Get the greater index.

Definition at line 68 of file edge_pair.hpp.

References greater.

21.13.3.4 size_t& mlpack::emst::EdgePair::Greater ( ) [inline]

Modify the greater index.

Definition at line 70 of file edge_pair.hpp.

References greater.

21.13.3.5 size_t mlpack::emst::EdgePair::Lesser ( ) const [inline]

Get the lesser index.

Definition at line 63 of file edge_pair.hpp.

References lesser.

21.13.3.6 size_t& mlpack::emst::EdgePair::Lesser ( ) [inline]

Modify the lesser index.

Definition at line 65 of file edge_pair.hpp.

References lesser.

21.13.4 Member Data Documentation

21.13.4.1 double mlpack::emst::EdgePair::distance [private]

Distance between two indices.

Definition at line 46 of file edge_pair.hpp.

Referenced by Distance().

21.13.4.2 size_t mlpack::emst::EdgePair::greater [private]

Greater index.

Definition at line 44 of file edge_pair.hpp.

Referenced by Greater().

21.13.4.3 size_t mlpack::emst::EdgePair::lesser [private]

Lesser index.

Definition at line 42 of file edge_pair.hpp.

Referenced by Lesser().

The documentation for this class was generated from the following file:

- src/mlpack/methods/emst/**edge_pair.hpp**

21.14 mlpack::emst::UnionFind Class Reference

A Union-Find data structure.

Public Member Functions

- **UnionFind** (const size_t **size**)

  Construct the object with the given size.
- ∼**UnionFind** ()

  Destroy the object (nothing to do).
- size_t **Find** (const size_t x)

  Returns the component containing an element.
- void **Union** (const size_t x, const size_t y)

  Union the components containing x and y.

Private Attributes

- arma::Col< size_t > **parent**
- arma::ivec **rank**
- size_t **size**

21.14.1 Detailed Description

A Union-Find data structure.

See Cormen, Rivest, & Stein for details. The structure tracks the components of a graph. Each point in the graph is initially in its own component. Calling Union(x, y) unites the components indexed by x and y. Find(x) returns the index of the component containing point x.

Definition at line 40 of file union_find.hpp.

21.14.2 Constructor & Destructor Documentation

21.14.2.1 mlpack::emst::UnionFind::UnionFind (const size_t *size*) [inline]

Construct the object with the given size.

Definition at line 49 of file union_find.hpp.

References parent, rank, and size.

21.14.2.2 mlpack::emst::UnionFind::∼UnionFind ( ) [inline]

Destroy the object (nothing to do).

Definition at line 59 of file union_find.hpp.

21.14.3 Member Function Documentation

21.14.3.1 size_t mlpack::emst::UnionFind::Find (const size_t *x*) [inline]

Returns the component containing an element.

Parameters

x	the component to be found

Returns

The index of the component containing x

Definition at line 67 of file union_find.hpp.

References parent.

Referenced by Union().

21.14.3.2 void mlpack::emst::UnionFind::Union (const size_t *x*, const size_t *y*) [inline]

Union the components containing x and y.

Parameters

x	one component
y	the other component

Definition at line 87 of file union_find.hpp.

References Find(), parent, and rank.

21.14.4 Member Data Documentation

21.14.4.1 arma::Col<size_t> mlpack::emst::UnionFind::parent [private]

Definition at line 44 of file union_find.hpp.

Referenced by Find(), Union(), and UnionFind().

21.14.4.2 arma::ivec mlpack::emst::UnionFind::rank [private]

Definition at line 45 of file union_find.hpp.

Referenced by Union(), and UnionFind().

21.14.4.3 size_t mlpack::emst::UnionFind::size [private]

Definition at line 43 of file union_find.hpp.

Referenced by UnionFind().

The documentation for this class was generated from the following file:

- src/mlpack/methods/emst/**union_find.hpp**

21.15 mlpack::fastmks::FastMKS< KernelType, TreeType > Class Template Reference

An implementation of fast exact max-kernel search.

Public Member Functions

- **FastMKS** (const arma::mat &**referenceSet**, const bool **single**=false, const bool **naive**=false)

  *Create the **FastMKS*** (p. *185) object using the reference set as the query set.*
- **FastMKS** (const arma::mat &**referenceSet**, const arma::mat &**querySet**, const bool **single**=false, const bool **naive**=false)

  *Create the **FastMKS*** (p. *185) object using separate reference and query sets.*
- **FastMKS** (const arma::mat &**referenceSet**, KernelType &kernel, const bool **single**=false, const bool **naive**=false)

  *Create the **FastMKS*** (p. *185) object using the reference set as the query set, and with an initialized kernel.*
- **FastMKS** (const arma::mat &**referenceSet**, const arma::mat &**querySet**, KernelType &kernel, const bool **single**=false, const bool **naive**=false)

  *Create the **FastMKS*** (p. *185) object using separate reference and query sets, and with an initialized kernel.*
- **FastMKS** (const arma::mat &**referenceSet**, TreeType ∗**referenceTree**, const bool **single**=false, const bool **naive**=false)

  *Create the **FastMKS*** (p. *185) object with an already-initialized tree built on the reference points.*
- **FastMKS** (const arma::mat &**referenceSet**, TreeType ∗**referenceTree**, const arma::mat &**querySet**, TreeType ∗**queryTree**, const bool **single**=false, const bool **naive**=false)

  *Create the **FastMKS*** (p. *185) object with already-initialized trees built on the reference and query points.*
- ∼**FastMKS** ()

  *Destructor for the **FastMKS*** (p. *185) object.*
- const **metric::IPMetric**
  < KernelType > & **Metric** () const

  Get the inner-product metric induced by the given kernel.
- **metric::IPMetric**< KernelType > & **Metric** ()

  Modify the inner-product metric induced by the given kernel.
- void **Search** (const size_t k, arma::Mat< size_t > &indices, arma::mat &products)

  Search for the maximum inner products of the query set (or if no query set was passed, the reference set is used).

Private Member Functions

- void **InsertNeighbor** (arma::Mat< size_t > &indices, arma::mat &products, const size_t queryIndex, const size_t pos, const size_t neighbor, const double distance)

  Utility function. Copied too many times from too many places.

Private Attributes

- **metric::IPMetric**< KernelType > **metric**

  The instantiated inner-product metric induced by the given kernel.
- bool **naive**

  If true, naive (brute-force) search is used.
- const arma::mat & **querySet**

  The query dataset.
- TreeType ∗ **queryTree**

  The tree built on the query dataset.
- const arma::mat & **referenceSet**

  The reference dataset.
- TreeType ∗ **referenceTree**

  The tree built on the reference dataset.

- bool **single**

  If true, single-tree search is used.

- bool **treeOwner**

  If true, this object created the trees and is responsible for them.

21.15.1 Detailed Description

template<typename KernelType, typename TreeType = tree::CoverTree<metric::IPMetric<KernelType>, tree::FirstPointIsRoot, FastMKSStat>>class mlpack::fastmks::FastMKS< KernelType, TreeType >

An implementation of fast exact max-kernel search.

Given a query dataset and a reference dataset (or optionally just a reference dataset which is also used as the query dataset), fast exact max-kernel search finds, for each point in the query dataset, the k points in the reference set with maximum kernel value K(p_q, p_r), where k is a specified parameter and K() is a Mercer kernel.

For more information, see the following paper.

```
@inproceedings{curtin2013fast,
  title={Fast Exact Max-Kernel Search},
  author={Curtin, Ryan R. and Ram, Parikshit and Gray, Alexander G.},
  booktitle={Proceedings of the 2013 SIAM International Conference on Data
      Mining (SDM 13)},
  year={2013}
}
```

This class allows specification of the type of kernel and also of the type of tree. **FastMKS** (p. 185) can be run on kernels that work on arbitrary objects – however, this only works with cover trees and other trees that are built only on points in the dataset (and not centroids of regions or anything like that).

Template Parameters

KernelType	Type of kernel to run **FastMKS** (p. 185) with.
TreeType	Type of tree to run **FastMKS** (p. 185) with; it must have metric IPMetric<KernelType>.

Definition at line 69 of file fastmks.hpp.

21.15.2 Constructor & Destructor Documentation

21.15.2.1 template<typename KernelType, typename TreeType = tree::CoverTree<metric::IPMetric<KernelType>, tree::FirstPointIsRoot, FastMKSStat>> mlpack::fastmks::FastMKS< KernelType, TreeType >::FastMKS (const arma::mat & *referenceSet*, const bool *single* = `false`, const bool *naive* = `false`)

Create the **FastMKS** (p. 185) object using the reference set as the query set.

Optionally, specify whether or not single-tree search or naive (brute-force) search should be used.

Parameters

referenceSet	Set of data to run **FastMKS** (p. 185) on.
single	Whether or not to run single-tree search.

naive	Whether or not to run brute-force (naive) search.

21.15.2.2 template<typename KernelType, typename TreeType = tree::CoverTree<metric::IPMetric<KernelType>, tree::FirstPointIsRoot, FastMKSStat>> mlpack::fastmks::FastMKS< KernelType, TreeType >::FastMKS (const arma::mat & *referenceSet*, const arma::mat & *querySet*, const bool *single* = false, const bool *naive* = false)

Create the **FastMKS** (p. 185) object using separate reference and query sets.

Optionally, specify whether or not single-tree search or naive (brute-force) search should be used.

Parameters

referenceSet	Reference set of data for **FastMKS** (p. 185).
querySet	Set of query points for **FastMKS** (p. 185).
single	Whether or not to run single-tree search.
naive	Whether or not to run brute-force (naive) search.

21.15.2.3 template<typename KernelType, typename TreeType = tree::CoverTree<metric::IPMetric<KernelType>, tree::FirstPointIsRoot, FastMKSStat>> mlpack::fastmks::FastMKS< KernelType, TreeType >::FastMKS (const arma::mat & *referenceSet*, KernelType & *kernel*, const bool *single* = false, const bool *naive* = false)

Create the **FastMKS** (p. 185) object using the reference set as the query set, and with an initialized kernel.

This is useful for when the kernel stores state. Optionally, specify whether or not single-tree search or naive (brute-force) search should be used.

Parameters

referenceSet	Reference set of data for **FastMKS** (p. 185).
kernel	Initialized kernel.
single	Whether or not to run single-tree search.
naive	Whether or not to run brute-force (naive) search.

21.15.2.4 template<typename KernelType, typename TreeType = tree::CoverTree<metric::IPMetric<KernelType>, tree::FirstPointIsRoot, FastMKSStat>> mlpack::fastmks::FastMKS< KernelType, TreeType >::FastMKS (const arma::mat & *referenceSet*, const arma::mat & *querySet*, KernelType & *kernel*, const bool *single* = false, const bool *naive* = false)

Create the **FastMKS** (p. 185) object using separate reference and query sets, and with an initialized kernel.

This is useful for when the kernel stores state. Optionally, specify whether or not single-tree search or naive (brute-force) search should be used.

Parameters

referenceSet	Reference set of data for **FastMKS** (p. 185).
querySet	Set of query points for **FastMKS** (p. 185).
kernel	Initialized kernel.

single	Whether or not to run single-tree search.
naive	Whether or not to run brute-force (naive) search.

21.15.2.5 template<typename KernelType, typename TreeType = tree::CoverTree<metric::IPMetric<KernelType>, tree::FirstPointIsRoot, FastMKSStat>> mlpack::fastmks::FastMKS< KernelType, TreeType >::FastMKS (const arma::mat & *referenceSet*, TreeType * *referenceTree*, const bool *single* = false, const bool *naive* = false)

Create the **FastMKS** (p. 185) object with an already-initialized tree built on the reference points.

Be sure that the tree is built with the metric type IPMetric<KernelType>. For this constructor, the reference set and the query set are the same points. Optionally, whether or not to run single-tree search or brute-force (naive) search can be specified.

Parameters

referenceSet	Reference set of data for **FastMKS** (p. 185).
referenceTree	Tree built on reference data.
single	Whether or not to run single-tree search.
naive	Whether or not to run brute-force (naive) search.

21.15.2.6 template<typename KernelType, typename TreeType = tree::CoverTree<metric::IPMetric<KernelType>, tree::FirstPointIsRoot, FastMKSStat>> mlpack::fastmks::FastMKS< KernelType, TreeType >::FastMKS (const arma::mat & *referenceSet*, TreeType * *referenceTree*, const arma::mat & *querySet*, TreeType * *queryTree*, const bool *single* = false, const bool *naive* = false)

Create the **FastMKS** (p. 185) object with already-initialized trees built on the reference and query points.

Be sure that the trees are built with the metric type IPMetric<KernelType>. Optionally, whether or not to run single-tree search or naive (brute-force) search can be specified.

Parameters

referenceSet	Reference set of data for **FastMKS** (p. 185).
referenceTree	Tree built on reference data.
querySet	Set of query points for **FastMKS** (p. 185).
queryTree	Tree built on query data.
single	Whether or not to use single-tree search.
naive	Whether or not to use naive (brute-force) search.

21.15.2.7 template<typename KernelType, typename TreeType = tree::CoverTree<metric::IPMetric<KernelType>, tree::FirstPointIsRoot, FastMKSStat>> mlpack::fastmks::FastMKS< KernelType, TreeType >::~FastMKS ( )

Destructor for the **FastMKS** (p. 185) object.

21.15.3 Member Function Documentation

21.15.3.1 template<typename KernelType, typename TreeType = tree::CoverTree<metric::IPMetric<KernelType>, tree::FirstPointIsRoot, FastMKSStat>> void mlpack::fastmks::FastMKS< KernelType, TreeType >::InsertNeighbor (arma::Mat< size_t > & *indices,* arma::mat & *products,* const size_t *queryIndex,* const size_t *pos,* const size_t *neighbor,* const double *distance*) [private]

Utility function. Copied too many times from too many places.

21.15.3.2 template<typename KernelType, typename TreeType = tree::CoverTree<metric::IPMetric<KernelType>, tree::FirstPointIsRoot, FastMKSStat>> const metric::IPMetric<KernelType>& mlpack::fastmks::FastMKS< KernelType, TreeType >::Metric ( ) const [inline]

Get the inner-product metric induced by the given kernel.

Definition at line 193 of file fastmks.hpp.

References mlpack::fastmks::FastMKS< KernelType, TreeType >::metric.

21.15.3.3 template<typename KernelType, typename TreeType = tree::CoverTree<metric::IPMetric<KernelType>, tree::FirstPointIsRoot, FastMKSStat>> metric::IPMetric<KernelType>& mlpack::fastmks::FastMKS< KernelType, TreeType >::Metric ( ) [inline]

Modify the inner-product metric induced by the given kernel.

Definition at line 195 of file fastmks.hpp.

References mlpack::fastmks::FastMKS< KernelType, TreeType >::metric.

21.15.3.4 template<typename KernelType, typename TreeType = tree::CoverTree<metric::IPMetric<KernelType>, tree::FirstPointIsRoot, FastMKSStat>> void mlpack::fastmks::FastMKS< KernelType, TreeType >::Search (const size_t *k,* arma::Mat< size_t > & *indices,* arma::mat & *products*)

Search for the maximum inner products of the query set (or if no query set was passed, the reference set is used).

The resulting maximum inner products are stored in the products matrix and the corresponding point indices are stores in the indices matrix. The results for each point in the query set are stored in the corresponding column of the indices and products matrices; for instance, the index of the point with maximum inner product to point 4 in the query set will be stored in row 0 and column 4 of the indices matrix.

Parameters

k	The number of maximum kernels to find.
indices	Matrix to store resulting indices of max-kernel search in.
products	Matrix to store resulting max-kernel values in.

21.15.4 Member Data Documentation

21.15.4.1 template<typename KernelType, typename TreeType = tree::CoverTree<metric::IPMetric<KernelType>, tree::FirstPointIsRoot, FastMKSStat>> metric::IPMetric<KernelType> mlpack::fastmks::FastMKS< KernelType, TreeType >::metric [private]

The instantiated inner-product metric induced by the given kernel.

Definition at line 218 of file fastmks.hpp.

Referenced by mlpack::fastmks::FastMKS< KernelType, TreeType >::Metric().

21.15.4.2 template<typename KernelType, typename TreeType = tree::CoverTree<metric::IPMetric<KernelType>, tree::FirstPointIsRoot, FastMKSStat>> bool mlpack::fastmks::FastMKS< KernelType, TreeType >::naive [private]

If true, naive (brute-force) search is used.

Definition at line 215 of file fastmks.hpp.

21.15.4.3 template<typename KernelType, typename TreeType = tree::CoverTree<metric::IPMetric<KernelType>, tree::FirstPointIsRoot, FastMKSStat>> const arma::mat& mlpack::fastmks::FastMKS< KernelType, TreeType >::querySet [private]

The query dataset.

Definition at line 201 of file fastmks.hpp.

21.15.4.4 template<typename KernelType, typename TreeType = tree::CoverTree<metric::IPMetric<KernelType>, tree::FirstPointIsRoot, FastMKSStat>> TreeType* mlpack::fastmks::FastMKS< KernelType, TreeType >::queryTree [private]

The tree built on the query dataset.

This is NULL if there is no query set.

Definition at line 207 of file fastmks.hpp.

21.15.4.5 template<typename KernelType, typename TreeType = tree::CoverTree<metric::IPMetric<KernelType>, tree::FirstPointIsRoot, FastMKSStat>> const arma::mat& mlpack::fastmks::FastMKS< KernelType, TreeType >::referenceSet [private]

The reference dataset.

Definition at line 199 of file fastmks.hpp.

21.15.4.6 template<typename KernelType, typename TreeType = tree::CoverTree<metric::IPMetric<KernelType>, tree::FirstPointIsRoot, FastMKSStat>> TreeType* mlpack::fastmks::FastMKS< KernelType, TreeType >::referenceTree [private]

The tree built on the reference dataset.

Definition at line 204 of file fastmks.hpp.

21.15.4.7 template<typename KernelType, typename TreeType = tree::CoverTree<metric::IPMetric<KernelType>, tree::FirstPointIsRoot, FastMKSStat>> bool mlpack::fastmks::FastMKS< KernelType, TreeType >::single [private]

If true, single-tree search is used.

Definition at line 213 of file fastmks.hpp.

21.15.4.8 template<typename KernelType, typename TreeType = tree::CoverTree<metric::IPMetric<KernelType>, tree::FirstPointIsRoot, FastMKSStat>> bool mlpack::fastmks::FastMKS< KernelType, TreeType >::treeOwner [private]

If true, this object created the trees and is responsible for them.

Definition at line 210 of file fastmks.hpp.

The documentation for this class was generated from the following file:

- src/mlpack/methods/fastmks/**fastmks.hpp**

21.16 mlpack::fastmks::FastMKSRules< KernelType, TreeType > Class Template Reference

The base case and pruning rules for **FastMKS** (p. 185) (fast max-kernel search).

Public Member Functions

- **FastMKSRules** (const arma::mat &**referenceSet**, const arma::mat &**querySet**, arma::Mat< size_t > &**indices**, arma::mat &**products**, KernelType &**kernel**)
- double **BaseCase** (const size_t queryIndex, const size_t referenceIndex)

  Compute the base case (kernel value) between two points.
- size_t **BaseCases** () const

  *Get the number of times **BaseCase()** (p. 193) was called.*
- size_t & **BaseCases** ()

  *Modify the number of times **BaseCase()** (p. 193) was called.*
- double **Rescore** (const size_t queryIndex, TreeType &referenceNode, const double oldScore) const

  Re-evaluate the score for recursion order.
- double **Rescore** (TreeType &queryNode, TreeType &referenceNode, const double oldScore) const

  Re-evaluate the score for recursion order.
- double **Score** (const size_t queryIndex, TreeType &referenceNode)

  Get the score for recursion order.
- double **Score** (TreeType &queryNode, TreeType &referenceNode)

  Get the score for recursion order.
- size_t **Scores** () const

  *Get the number of times **Score()** (p. 195) was called.*
- size_t & **Scores** ()

  *Modify the number of times **Score()** (p. 195) was called.*

Private Member Functions

- double **CalculateBound** (TreeType &queryNode) const

  Calculate the bound for a given query node.
- void **InsertNeighbor** (const size_t queryIndex, const size_t pos, const size_t neighbor, const double distance)

  Utility function to insert neighbor into list of results.

Private Attributes

- size_t **baseCases**

  For benchmarking.
- arma::Mat< size_t > & **indices**

  The indices of the maximum kernel results.
- KernelType & **kernel**

  The instantiated kernel.
- double **lastKernel**

  *The last kernel evaluation resulting from **BaseCase()** (p. 193).*
- size_t **lastQueryIndex**

  *The last query index **BaseCase()** (p. 193) was called with.*
- size_t **lastReferenceIndex**

  *The last reference index **BaseCase()** (p. 193) was called with.*
- arma::mat & **products**

  The maximum kernels.
- arma::vec **queryKernels**

  Cached query set self-kernels (|| q || for each q).
- const arma::mat & **querySet**

  The query dataset.
- arma::vec **referenceKernels**

  Cached reference set self-kernels (|| r || for each r).
- const arma::mat & **referenceSet**

  The reference dataset.
- size_t **scores**

  For benchmarking.

21.16.1 Detailed Description

template<typename KernelType, typename TreeType>class mlpack::fastmks::FastMKSRules< KernelType, TreeType >

The base case and pruning rules for **FastMKS** (p. 185) (fast max-kernel search).

Definition at line 35 of file fastmks_rules.hpp.

21.16.2 Constructor & Destructor Documentation

21.16.2.1 template<typename KernelType , typename TreeType > mlpack::fastmks::FastMKSRules< KernelType, TreeType >::FastMKSRules (const arma::mat & *referenceSet,* const arma::mat & *querySet,* arma::Mat< size_t > & *indices,* arma::mat & *products,* KernelType & *kernel*)

21.16.3 Member Function Documentation

21.16.3.1 template<typename KernelType , typename TreeType > double mlpack::fastmks::FastMKSRules< KernelType, TreeType >::BaseCase (const size_t *queryIndex,* const size_t *referenceIndex*)

Compute the base case (kernel value) between two points.

21.16.3.2 template<typename KernelType , typename TreeType > size_t mlpack::fastmks::FastMKSRules< KernelType, TreeType >::BaseCases ( ) const [inline]

Get the number of times **BaseCase()** (p. 193) was called.

Definition at line 98 of file fastmks_rules.hpp.

References mlpack::fastmks::FastMKSRules< KernelType, TreeType >::baseCases.

21.16.3.3 template<typename KernelType , typename TreeType > size_t& mlpack::fastmks::FastMKSRules< KernelType, TreeType >::BaseCases ( ) [inline]

Modify the number of times **BaseCase()** (p. 193) was called.

Definition at line 100 of file fastmks_rules.hpp.

References mlpack::fastmks::FastMKSRules< KernelType, TreeType >::baseCases.

21.16.3.4 template<typename KernelType , typename TreeType > double mlpack::fastmks::FastMKSRules< KernelType, TreeType >::CalculateBound (TreeType & *queryNode*) const [private]

Calculate the bound for a given query node.

21.16.3.5 template<typename KernelType , typename TreeType > void mlpack::fastmks::FastMKSRules< KernelType, TreeType >::InsertNeighbor (const size_t *queryIndex,* const size_t *pos,* const size_t *neighbor,* const double *distance*) [private]

Utility function to insert neighbor into list of results.

21.16.3.6 template<typename KernelType , typename TreeType > double mlpack::fastmks::FastMKSRules< KernelType, TreeType >::Rescore (const size_t *queryIndex,* TreeType & *referenceNode,* const double *oldScore*) const

Re-evaluate the score for recursion order.

A low score indicates priority for recursion, while DBL_MAX indicates that a node should not be recursed into at all (it should be pruned). This is used when the score has already been calculated, but another recursion may have modified the bounds for pruning. So the old score is checked against the new pruning bound.

Parameters

queryIndex	Index of query point.
referenceNode	Candidate node to be recursed into.
oldScore	Old score produced by **Score()** (p. 195) (or **Rescore()** (p. 194)).

21.16.3.7 template<typename KernelType , typename TreeType > double mlpack::fastmks::FastMKSRules< KernelType, TreeType >::Rescore (TreeType & *queryNode,* TreeType & *referenceNode,* const double *oldScore*) const

Re-evaluate the score for recursion order.

A low score indicates priority for recursion, while DBL_MAX indicates that a node should not be recursed into at all (it should be pruned). This is used when the score has already been calculated, but another recursion may have modified the bounds for pruning. So the old score is checked against the new pruning bound.

Parameters

queryNode	Candidate query node to be recursed into.
referenceNode	Candidate reference node to be recursed into.
oldScore	Old score produced by **Score()** (p. 195) (or **Rescore()** (p. 194)).

21.16.3.8 template<typename KernelType , typename TreeType > double mlpack::fastmks::FastMKSRules< KernelType, TreeType >::Score (const size_t *queryIndex,* TreeType & *referenceNode*)

Get the score for recursion order.

A low score indicates priority for recursion, while DBL_MAX indicates that the node should not be recursed into at all (it should be pruned).

Parameters

queryIndex	Index of query point.
referenceNode	Candidate to be recursed into.

21.16.3.9 template<typename KernelType , typename TreeType > double mlpack::fastmks::FastMKSRules< KernelType, TreeType >::Score (TreeType & *queryNode,* TreeType & *referenceNode*)

Get the score for recursion order.

A low score indicates priority for recursion, while DBL_MAX indicates that the node should not be recursed into at all (it should be pruned).

Parameters

queryNode	Candidate query node to be recursed into.
referenceNode	Candidate reference node to be recursed into.

21.16.3.10 template<typename KernelType , typename TreeType > size_t mlpack::fastmks::FastMKSRules< KernelType, TreeType >::Scores ( ) const [inline]

Get the number of times **Score()** (p. 195) was called.

Definition at line 103 of file fastmks_rules.hpp.

References mlpack::fastmks::FastMKSRules< KernelType, TreeType >::scores.

21.16.3.11 template<typename KernelType , typename TreeType > size_t& mlpack::fastmks::FastMKSRules< KernelType, TreeType >::Scores ( ) [inline]

Modify the number of times **Score()** (p. 195) was called.

Definition at line 105 of file fastmks_rules.hpp.

References mlpack::fastmks::FastMKSRules< KernelType, TreeType >::scores.

21.16.4 Member Data Documentation

21.16.4.1 template<typename KernelType , typename TreeType > size_t mlpack::fastmks::FastMKSRules< KernelType, TreeType >::baseCases [private]

For benchmarking.

Definition at line 143 of file fastmks_rules.hpp.

Referenced by mlpack::fastmks::FastMKSRules< KernelType, TreeType >::BaseCases().

21.16.4.2 template<typename KernelType , typename TreeType > arma::Mat<size_t>& mlpack::fastmks::FastMKSRules< KernelType, TreeType >::indices [private]

The indices of the maximum kernel results.

Definition at line 114 of file fastmks_rules.hpp.

21.16.4.3 template<typename KernelType , typename TreeType > KernelType& mlpack::fastmks::FastMKSRules< KernelType, TreeType >::kernel [private]

The instantiated kernel.

Definition at line 124 of file fastmks_rules.hpp.

21.16.4.4 template<typename KernelType , typename TreeType > double mlpack::fastmks::FastMKSRules< KernelType, TreeType >::lastKernel [private]

The last kernel evaluation resulting from **BaseCase()** (p. 193).

Definition at line 131 of file fastmks_rules.hpp.

21.16.4.5 template<typename KernelType , typename TreeType > size_t mlpack::fastmks::FastMKSRules< KernelType, TreeType >::lastQueryIndex [private]

The last query index **BaseCase()** (p. 193) was called with.

Definition at line 127 of file fastmks_rules.hpp.

21.16.4.6 template<typename KernelType , typename TreeType > size_t mlpack::fastmks::FastMKSRules< KernelType, TreeType >::lastReferenceIndex [private]

The last reference index **BaseCase()** (p. 193) was called with.

Definition at line 129 of file fastmks_rules.hpp.

21.16.4.7 template<typename KernelType , typename TreeType > arma::mat& mlpack::fastmks::FastMKSRules< KernelType, TreeType >::products [private]

The maximum kernels.

Definition at line 116 of file fastmks_rules.hpp.

21.16.4.8 template<typename KernelType , typename TreeType > arma::vec mlpack::fastmks::FastMKSRules< KernelType, TreeType >::queryKernels [private]

Cached query set self-kernels (|| q || for each q).

Definition at line 119 of file fastmks_rules.hpp.

21.16.4.9 template<typename KernelType , typename TreeType > const arma::mat& mlpack::fastmks::FastMKSRules< KernelType, TreeType >::querySet [private]

The query dataset.

Definition at line 111 of file fastmks_rules.hpp.

21.16.4.10 template<typename KernelType , typename TreeType > arma::vec mlpack::fastmks::FastMKSRules< KernelType, TreeType >::referenceKernels [private]

Cached reference set self-kernels (|| r || for each r).

Definition at line 121 of file fastmks_rules.hpp.

21.16.4.11 template<typename KernelType , typename TreeType > const arma::mat& mlpack::fastmks::FastMKSRules< KernelType, TreeType >::referenceSet [private]

The reference dataset.

Definition at line 109 of file fastmks_rules.hpp.

21.16.4.12 template<typename KernelType , typename TreeType > size_t mlpack::fastmks::FastMKSRules< KernelType, TreeType >::scores [private]

For benchmarking.

Definition at line 145 of file fastmks_rules.hpp.

Referenced by mlpack::fastmks::FastMKSRules< KernelType, TreeType >::Scores().

The documentation for this class was generated from the following file:

- src/mlpack/methods/fastmks/**fastmks_rules.hpp**

21.17 mlpack::fastmks::FastMKSStat Class Reference

The statistic used in trees with **FastMKS** (p. 185).

Public Member Functions

- **FastMKSStat** ()

  Default initialization.

- template<typename TreeType >
  FastMKSStat (const TreeType &node)

Initialize this statistic for the given tree node.

- double **Bound** () const

  Get the bound.
- double & **Bound** ()

  Modify the bound.
- double **LastKernel** () const

  Get the last kernel evaluation.
- double & **LastKernel** ()

  Modify the last kernel evaluation.
- void * **LastKernelNode** () const

  Get the address of the node corresponding to the last distance evaluation.
- void *& **LastKernelNode** ()

  Modify the address of the node corresponding to the last distance evaluation.
- double **SelfKernel** () const

  Get the self-kernel.
- double & **SelfKernel** ()

  Modify the self-kernel.

Private Attributes

- double **bound**

  The bound for pruning.
- double **lastKernel**

  The last kernel evaluation.
- void * **lastKernelNode**

  The node corresponding to the last kernel evaluation.
- double **selfKernel**

  The self-kernel evaluation: sqrt(K(centroid, centroid)).

21.17.1 Detailed Description

The statistic used in trees with **FastMKS** (p. 185).

This stores both the bound and the self-kernels for each node in the tree.

Definition at line 35 of file fastmks_stat.hpp.

21.17.2 Constructor & Destructor Documentation

21.17.2.1 mlpack::fastmks::FastMKSStat::FastMKSStat ( ) [inline]

Default initialization.

Definition at line 41 of file fastmks_stat.hpp.

21.17.2.2 template<typename TreeType > mlpack::fastmks::FastMKSStat::FastMKSStat (const TreeType & *node*) [inline]

Initialize this statistic for the given tree node.

The TreeType's metric better be IPMetric with some kernel type (that is, Metric().Kernel() must exist).

Parameters

node	Node that this statistic is built for.

Definition at line 56 of file fastmks_stat.hpp.

References selfKernel.

21.17.3 Member Function Documentation

21.17.3.1 double mlpack::fastmks::FastMKSStat::Bound ( ) const [inline]

Get the bound.

Definition at line 96 of file fastmks_stat.hpp.

References bound.

21.17.3.2 double& mlpack::fastmks::FastMKSStat::Bound ( ) [inline]

Modify the bound.

Definition at line 98 of file fastmks_stat.hpp.

References bound.

21.17.3.3 double mlpack::fastmks::FastMKSStat::LastKernel ( ) const [inline]

Get the last kernel evaluation.

Definition at line 101 of file fastmks_stat.hpp.

References lastKernel.

21.17.3.4 double& mlpack::fastmks::FastMKSStat::LastKernel ( ) [inline]

Modify the last kernel evaluation.

Definition at line 103 of file fastmks_stat.hpp.

References lastKernel.

21.17.3.5 void* mlpack::fastmks::FastMKSStat::LastKernelNode ( ) const [inline]

Get the address of the node corresponding to the last distance evaluation.

Definition at line 106 of file fastmks_stat.hpp.

References lastKernelNode.

21.17.3.6 void*& mlpack::fastmks::FastMKSStat::LastKernelNode ( ) [inline]

Modify the address of the node corresponding to the last distance evaluation.

Definition at line 109 of file fastmks_stat.hpp.

References lastKernelNode.

21.17.3.7 double mlpack::fastmks::FastMKSStat::SelfKernel ( ) const [inline]

Get the self-kernel.

Definition at line 91 of file fastmks_stat.hpp.

References selfKernel.

21.17.3.8 double& mlpack::fastmks::FastMKSStat::SelfKernel ( ) [inline]

Modify the self-kernel.

Definition at line 93 of file fastmks_stat.hpp.

References selfKernel.

21.17.4 Member Data Documentation

21.17.4.1 double mlpack::fastmks::FastMKSStat::bound [private]

The bound for pruning.

Definition at line 113 of file fastmks_stat.hpp.

Referenced by Bound().

21.17.4.2 double mlpack::fastmks::FastMKSStat::lastKernel [private]

The last kernel evaluation.

Definition at line 119 of file fastmks_stat.hpp.

Referenced by LastKernel().

21.17.4.3 void* mlpack::fastmks::FastMKSStat::lastKernelNode [private]

The node corresponding to the last kernel evaluation.

This has to be void otherwise we get recursive template arguments.

Definition at line 123 of file fastmks_stat.hpp.

Referenced by LastKernelNode().

21.17.4.4 double mlpack::fastmks::FastMKSStat::selfKernel [private]

The self-kernel evaluation: sqrt(K(centroid, centroid)).

Definition at line 116 of file fastmks_stat.hpp.

Referenced by FastMKSStat(), and SelfKernel().

The documentation for this class was generated from the following file:

- src/mlpack/methods/fastmks/**fastmks_stat.hpp**

21.18 mlpack::gmm::DiagonalConstraint Class Reference

Force a covariance matrix to be diagonal.

Static Public Member Functions

- static void **ApplyConstraint** (arma::mat &covariance)

  Force a covariance matrix to be diagonal.

21.18.1 Detailed Description

Force a covariance matrix to be diagonal.

Definition at line 33 of file diagonal_constraint.hpp.

21.18.2 Member Function Documentation

21.18.2.1 static void mlpack::gmm::DiagonalConstraint::ApplyConstraint (arma::mat & *covariance*) [inline], [static]

Force a covariance matrix to be diagonal.

Definition at line 37 of file diagonal_constraint.hpp.

The documentation for this class was generated from the following file:

- src/mlpack/methods/gmm/**diagonal_constraint.hpp**

21.19 mlpack::gmm::EigenvalueRatioConstraint Class Reference

Given a vector of eigenvalue ratios, ensure that the covariance matrix always has those eigenvalue ratios.

Public Member Functions

- **EigenvalueRatioConstraint** (const arma::vec &**ratios**)

  *Create the **EigenvalueRatioConstraint** (p. 201) object with the given vector of eigenvalue ratios.*

- void **ApplyConstraint** (arma::mat &covariance) const

  Apply the eigenvalue ratio constraint to the given covariance matrix.

Private Attributes

- const arma::vec & **ratios**

  Ratios for eigenvalues.

21.19.1 Detailed Description

Given a vector of eigenvalue ratios, ensure that the covariance matrix always has those eigenvalue ratios.

When you create this object, make sure that the vector of ratios that you pass does not go out of scope, because this object holds a reference to that vector instead of copying it.

Definition at line 36 of file eigenvalue_ratio_constraint.hpp.

21.19.2 Constructor & Destructor Documentation

21.19.2.1 mlpack::gmm::EigenvalueRatioConstraint::EigenvalueRatioConstraint (const arma::vec & *ratios*) [inline]

Create the **EigenvalueRatioConstraint** (p. 201) object with the given vector of eigenvalue ratios.

These ratios are with respect to the first eigenvalue, which is the largest eigenvalue, so the first element of the vector should be 1. In addition, all other elements should be less than or equal to 1.

Definition at line 45 of file eigenvalue_ratio_constraint.hpp.

References mlpack::Log::Fatal, and mlpack::Log::Warn.

21.19.3 Member Function Documentation

21.19.3.1 void mlpack::gmm::EigenvalueRatioConstraint::ApplyConstraint (arma::mat & *covariance*) const [inline]

Apply the eigenvalue ratio constraint to the given covariance matrix.

Definition at line 70 of file eigenvalue_ratio_constraint.hpp.

References ratios.

21.19.4 Member Data Documentation

21.19.4.1 const arma::vec& mlpack::gmm::EigenvalueRatioConstraint::ratios [private]

Ratios for eigenvalues.

Definition at line 89 of file eigenvalue_ratio_constraint.hpp.

Referenced by ApplyConstraint().

The documentation for this class was generated from the following file:

- src/mlpack/methods/gmm/**eigenvalue_ratio_constraint.hpp**

21.20 mlpack::gmm::EMFit< InitialClusteringType, CovarianceConstraintPolicy > Class Template Reference

This class contains methods which can fit a **GMM** (p. 208) to observations using the EM algorithm.

Public Member Functions

- **EMFit** (const size_t **maxIterations**=300, const double **tolerance**=1e-10, InitialClusteringType clusterer=InitialClusteringType(), CovarianceConstraintPolicy constraint=CovarianceConstraintPolicy())

  *Construct the **EMFit** (p. 202) object, optionally passing an InitialClusteringType object (just in case it needs to store state).*
- const InitialClusteringType & **Clusterer** () const

  Get the clusterer.
- InitialClusteringType & **Clusterer** ()

  Modify the clusterer.
- const CovarianceConstraintPolicy & **Constraint** () const

  Get the covariance constraint policy class.
- CovarianceConstraintPolicy & **Constraint** ()

  Modify the covariance constraint policy class.
- void **Estimate** (const arma::mat &observations, std::vector< arma::vec > &means, std::vector< arma::mat > &covariances, arma::vec &weights, const bool useInitialModel=false)

  *Fit the observations to a Gaussian mixture model (**GMM** (p. 208)) using the EM algorithm.*
- void **Estimate** (const arma::mat &observations, const arma::vec &probabilities, std::vector< arma::vec > &means, std::vector< arma::mat > &covariances, arma::vec &weights, const bool useInitialModel=false)

  *Fit the observations to a Gaussian mixture model (**GMM** (p. 208)) using the EM algorithm, taking into account the probabilities of each point being from this mixture.*
- size_t **MaxIterations** () const

  Get the maximum number of iterations of the EM algorithm.
- size_t & **MaxIterations** ()

  Modify the maximum number of iterations of the EM algorithm.
- double **Tolerance** () const

  Get the tolerance for the convergence of the EM algorithm.
- double & **Tolerance** ()

  Modify the tolerance for the convergence of the EM algorithm.

Private Member Functions

- void **InitialClustering** (const arma::mat &observations, std::vector< arma::vec > &means, std::vector< arma::mat > &covariances, arma::vec &weights)

  Run the clusterer, and then turn the cluster assignments into Gaussians.
- double **LogLikelihood** (const arma::mat &data, const std::vector< arma::vec > &means, const std::vector< arma::mat > &covariances, const arma::vec &weights) const

  Calculate the log-likelihood of a model.

Private Attributes

- InitialClusteringType **clusterer**

  Object which will perform the clustering.
- CovarianceConstraintPolicy **constraint**

  Object which applies constraints to the covariance matrix.
- size_t **maxIterations**

  Maximum iterations of EM algorithm.
- double **tolerance**

  Tolerance for convergence of EM.

21.20.1 Detailed Description

template<typename InitialClusteringType = kmeans::KMeans<>, typename CovarianceConstraintPolicy = PositiveDefiniteConstraint>class mlpack::gmm::EMFit< InitialClusteringType, CovarianceConstraintPolicy >

This class contains methods which can fit a **GMM** (p. 208) to observations using the EM algorithm.

It requires an initial clustering mechanism, which is by default the KMeans algorithm. The clustering mechanism must implement the following method:

- void Cluster(const arma::mat& observations, const size_t clusters, arma::Col<size_t>& assignments);

This method should create 'clusters' clusters, and return the assignment of each point to a cluster.

Definition at line 51 of file em_fit.hpp.

21.20.2 Constructor & Destructor Documentation

21.20.2.1 template<typename InitialClusteringType = kmeans::KMeans<>, typename CovarianceConstraintPolicy = PositiveDefiniteConstraint> mlpack::gmm::EMFit< InitialClusteringType, CovarianceConstraintPolicy >::EMFit (const size_t *maxIterations* = 300, const double *tolerance* = 1e-10, InitialClusteringType *clusterer* = InitialClusteringType(), CovarianceConstraintPolicy *constraint* = CovarianceConstraintPolicy())

Construct the **EMFit** (p. 202) object, optionally passing an InitialClusteringType object (just in case it needs to store state).

Setting the maximum number of iterations to 0 means that the EM algorithm will iterate until convergence (with the given tolerance).

The parameter forcePositive controls whether or not the covariance matrices are checked for positive definiteness at each iteration. This could be a time-consuming task, so, if you know your data is well-behaved, you can set it to false and save some runtime.

Parameters

maxIterations	Maximum number of iterations for EM.
tolerance	Log-likelihood tolerance required for convergence.
forcePositive	Check for positive-definiteness of each covariance matrix at each iteration.
clusterer	Object which will perform the initial clustering.

21.20.3 Member Function Documentation

21.20.3.1 template<typename InitialClusteringType = kmeans::KMeans<>, typename CovarianceConstraintPolicy = PositiveDefiniteConstraint> const InitialClusteringType& mlpack::gmm::EMFit< InitialClusteringType, CovarianceConstraintPolicy >::Clusterer ( ) const [inline]

Get the clusterer.

Definition at line 122 of file em_fit.hpp.

References mlpack::gmm::EMFit< InitialClusteringType, CovarianceConstraintPolicy >::clusterer.

21.20.3.2 template<typename InitialClusteringType = kmeans::KMeans<>, typename CovarianceConstraintPolicy = PositiveDefiniteConstraint> InitialClusteringType& mlpack::gmm::EMFit< InitialClusteringType, CovarianceConstraintPolicy >::Clusterer ( ) [inline]

Modify the clusterer.

Definition at line 124 of file em_fit.hpp.

References mlpack::gmm::EMFit< InitialClusteringType, CovarianceConstraintPolicy >::clusterer.

21.20.3.3 template<typename InitialClusteringType = kmeans::KMeans<>, typename CovarianceConstraintPolicy = PositiveDefiniteConstraint> const CovarianceConstraintPolicy& mlpack::gmm::EMFit< InitialClusteringType, CovarianceConstraintPolicy >::Constraint ( ) const [inline]

Get the covariance constraint policy class.

Definition at line 127 of file em_fit.hpp.

References mlpack::gmm::EMFit< InitialClusteringType, CovarianceConstraintPolicy >::constraint.

21.20.3.4 template<typename InitialClusteringType = kmeans::KMeans<>, typename CovarianceConstraintPolicy = PositiveDefiniteConstraint> CovarianceConstraintPolicy& mlpack::gmm::EMFit< InitialClusteringType, CovarianceConstraintPolicy >::Constraint ( ) [inline]

Modify the covariance constraint policy class.

Definition at line 129 of file em_fit.hpp.

References mlpack::gmm::EMFit< InitialClusteringType, CovarianceConstraintPolicy >::constraint.

21.20.3.5 template<typename InitialClusteringType = kmeans::KMeans<>, typename CovarianceConstraintPolicy = PositiveDefiniteConstraint> void mlpack::gmm::EMFit< InitialClusteringType, CovarianceConstraintPolicy >::Estimate (const arma::mat & *observations,* std::vector< arma::vec > & *means,* std::vector< arma::mat > & *covariances,* arma::vec & *weights,* const bool *useInitialModel* = false)

Fit the observations to a Gaussian mixture model (**GMM** (p. 208)) using the EM algorithm.

The size of the vectors (indicating the number of components) must already be set. Optionally, if useInitialModel is set to true, then the model given in the means, covariances, and weights parameters is used as the initial model, instead of using the InitialClusteringType::Cluster() option.

Parameters

observations	List of observations to train on.
means	Vector to store trained means in.
covariances	Vector to store trained covariances in.
weights	Vector to store a priori weights in.
useInitialModel	If true, the given model is used for the initial clustering.

21.20.3.6 template<typename InitialClusteringType = kmeans::KMeans<>, typename CovarianceConstraintPolicy = PositiveDefiniteConstraint> void mlpack::gmm::EMFit< InitialClusteringType, CovarianceConstraintPolicy >::Estimate (const arma::mat & *observations,* const arma::vec & *probabilities,* std::vector< arma::vec > & *means,* std::vector< arma::mat > & *covariances,* arma::vec & *weights,* const bool *useInitialModel* = false)

Fit the observations to a Gaussian mixture model (**GMM** (p. 208)) using the EM algorithm, taking into account the probabilities of each point being from this mixture.

The size of the vectors (indicating the number of components) must already be set. Optionally, if useInitialModel is set to true, then the model given in the means, covariances, and weights parameters is used as the initial model, instead of using the InitialClusteringType::Cluster() option.

Parameters

observations	List of observations to train on.
probabilities	Probability of each point being from this model.
means	Vector to store trained means in.
covariances	Vector to store trained covariances in.
weights	Vector to store a priori weights in.
useInitialModel	If true, the given model is used for the initial clustering.

21.20.3.7 template<typename InitialClusteringType = kmeans::KMeans<>, typename CovarianceConstraintPolicy = PositiveDefiniteConstraint> void mlpack::gmm::EMFit< InitialClusteringType, CovarianceConstraintPolicy >::InitialClustering (const arma::mat & *observations,* std::vector< arma::vec > & *means,* std::vector< arma::mat > & *covariances,* arma::vec & *weights*) [private]

Run the clusterer, and then turn the cluster assignments into Gaussians.

This is a helper function for both overloads of **Estimate()** (p. 205). The vectors must be already set to the number of clusters.

Parameters

observations	List of observations.
means	Vector to store means in.
covariances	Vector to store covariances in.
weights	Vector to store a priori weights in.

21.20.3.8 template<typename InitialClusteringType = kmeans::KMeans<>, typename CovarianceConstraintPolicy = PositiveDefiniteConstraint> double mlpack::gmm::EMFit< InitialClusteringType, CovarianceConstraintPolicy >::LogLikelihood (const arma::mat & *data,* const std::vector< arma::vec > & *means,* const std::vector< arma::mat > & *covariances,* const arma::vec & *weights*) const [private]

Calculate the log-likelihood of a model.

Yes, this is reimplemented in the **GMM** (p. 208) code. Intuition suggests that the log-likelihood is not the best way to determine if the EM algorithm has converged.

Parameters

data	Data matrix.

means	Vector of means.
covariances	Vector of covariance matrices.
weights	Vector of a priori weights.

21.20.3.9 template<typename InitialClusteringType = kmeans::KMeans<>, typename CovarianceConstraintPolicy = PositiveDefiniteConstraint> size_t mlpack::gmm::EMFit< InitialClusteringType, CovarianceConstraintPolicy >::MaxIterations ( ) const [inline]

Get the maximum number of iterations of the EM algorithm.

Definition at line 132 of file em_fit.hpp.

References mlpack::gmm::EMFit< InitialClusteringType, CovarianceConstraintPolicy >::maxIterations.

21.20.3.10 template<typename InitialClusteringType = kmeans::KMeans<>, typename CovarianceConstraintPolicy = PositiveDefiniteConstraint> size_t& mlpack::gmm::EMFit< InitialClusteringType, CovarianceConstraintPolicy >::MaxIterations ( ) [inline]

Modify the maximum number of iterations of the EM algorithm.

Definition at line 134 of file em_fit.hpp.

References mlpack::gmm::EMFit< InitialClusteringType, CovarianceConstraintPolicy >::maxIterations.

21.20.3.11 template<typename InitialClusteringType = kmeans::KMeans<>, typename CovarianceConstraintPolicy = PositiveDefiniteConstraint> double mlpack::gmm::EMFit< InitialClusteringType, CovarianceConstraintPolicy >::Tolerance ( ) const [inline]

Get the tolerance for the convergence of the EM algorithm.

Definition at line 137 of file em_fit.hpp.

References mlpack::gmm::EMFit< InitialClusteringType, CovarianceConstraintPolicy >::tolerance.

21.20.3.12 template<typename InitialClusteringType = kmeans::KMeans<>, typename CovarianceConstraintPolicy = PositiveDefiniteConstraint> double& mlpack::gmm::EMFit< InitialClusteringType, CovarianceConstraintPolicy >::Tolerance ( ) [inline]

Modify the tolerance for the convergence of the EM algorithm.

Definition at line 139 of file em_fit.hpp.

References mlpack::gmm::EMFit< InitialClusteringType, CovarianceConstraintPolicy >::tolerance.

21.20.4 Member Data Documentation

21.20.4.1 template<typename InitialClusteringType = kmeans::KMeans<>, typename CovarianceConstraintPolicy = PositiveDefiniteConstraint> InitialClusteringType mlpack::gmm::EMFit< InitialClusteringType, CovarianceConstraintPolicy >::clusterer [private]

Object which will perform the clustering.

Definition at line 177 of file em_fit.hpp.

Referenced by mlpack::gmm::EMFit< InitialClusteringType, CovarianceConstraintPolicy >::Clusterer().

21.20.4.2 template<typename InitialClusteringType = kmeans::KMeans<>, typename CovarianceConstraintPolicy = PositiveDefiniteConstraint> CovarianceConstraintPolicy mlpack::gmm::EMFit< InitialClusteringType, CovarianceConstraintPolicy >::constraint [private]

Object which applies constraints to the covariance matrix.

Definition at line 179 of file em_fit.hpp.

Referenced by mlpack::gmm::EMFit< InitialClusteringType, CovarianceConstraintPolicy >::Constraint().

21.20.4.3 template<typename InitialClusteringType = kmeans::KMeans<>, typename CovarianceConstraintPolicy = PositiveDefiniteConstraint> size_t mlpack::gmm::EMFit< InitialClusteringType, CovarianceConstraintPolicy >::maxIterations [private]

Maximum iterations of EM algorithm.

Definition at line 173 of file em_fit.hpp.

Referenced by mlpack::gmm::EMFit< InitialClusteringType, CovarianceConstraintPolicy >::MaxIterations().

21.20.4.4 template<typename InitialClusteringType = kmeans::KMeans<>, typename CovarianceConstraintPolicy = PositiveDefiniteConstraint> double mlpack::gmm::EMFit< InitialClusteringType, CovarianceConstraintPolicy >::tolerance [private]

Tolerance for convergence of EM.

Definition at line 175 of file em_fit.hpp.

Referenced by mlpack::gmm::EMFit< InitialClusteringType, CovarianceConstraintPolicy >::Tolerance().

The documentation for this class was generated from the following file:

- src/mlpack/methods/gmm/**em_fit.hpp**

21.21 mlpack::gmm::GMM< FittingType > Class Template Reference

A Gaussian Mixture Model (**GMM** (p. 208)).

Public Member Functions

- **GMM** ()

  Create an empty Gaussian Mixture Model, with zero gaussians.
- **GMM** (const size_t **gaussians**, const size_t **dimensionality**)

  *Create a **GMM** (p. 208) with the given number of Gaussians, each of which have the specified dimensionality.*
- **GMM** (const size_t **gaussians**, const size_t **dimensionality**, FittingType &**fitter**)

  *Create a **GMM** (p. 208) with the given number of Gaussians, each of which have the specified dimensionality.*
- **GMM** (const std::vector< arma::vec > &**means**, const std::vector< arma::mat > &**covariances**, const arma::vec &**weights**)

  *Create a **GMM** (p. 208) with the given means, covariances, and weights.*
- **GMM** (const std::vector< arma::vec > &**means**, const std::vector< arma::mat > &**covariances**, const arma::vec &**weights**, FittingType &**fitter**)

  *Create a **GMM** (p. 208) with the given means, covariances, and weights, and use the given initialized FittingType class.*

- template<typename OtherFittingType >
  GMM (const **GMM**< OtherFittingType > &other)

  Copy constructor for GMMs which use different fitting types.
- **GMM** (const **GMM** &other)

  Copy constructor for GMMs using the same fitting type.
- void **Classify** (const arma::mat &observations, arma::Col< size_t > &labels) const

  *Classify the given observations as being from an individual component in this **GMM** (p. 208).*
- const std::vector< arma::mat > & **Covariances** () const

  Return a const reference to the vector of covariance matrices (sigma).
- std::vector< arma::mat > & **Covariances** ()

  Return a reference to the vector of covariance matrices (sigma).
- size_t **Dimensionality** () const

  Return the dimensionality of the model.
- size_t & **Dimensionality** ()

  Modify the dimensionality of the model.
- double **Estimate** (const arma::mat &observations, const size_t trials=1, const bool useExistingModel=false)

  Estimate the probability distribution directly from the given observations, using the given algorithm in the FittingType class to fit the data.
- double **Estimate** (const arma::mat &observations, const arma::vec &probabilities, const size_t trials=1, const bool useExistingModel=false)

  Estimate the probability distribution directly from the given observations, taking into account the probability of each observation actually being from this distribution, and using the given algorithm in the FittingType class to fit the data.
- const FittingType & **Fitter** () const

  Return a const reference to the fitting type.
- FittingType & **Fitter** ()

  Return a reference to the fitting type.
- size_t **Gaussians** () const

  Return the number of gaussians in the model.
- size_t & **Gaussians** ()

  Modify the number of gaussians in the model.
- void **Load** (const std::string &filename)

  *Load a **GMM** (p. 208) from an XML file.*
- const std::vector< arma::vec > & **Means** () const

  Return a const reference to the vector of means (mu).
- std::vector< arma::vec > & **Means** ()

  Return a reference to the vector of means (mu).
- template<typename OtherFittingType >
  GMM & **operator=** (const **GMM**< OtherFittingType > &other)

  Copy operator for GMMs which use different fitting types.
- **GMM** & **operator=** (const **GMM** &other)

  Copy operator for GMMs which use the same fitting type.
- double **Probability** (const arma::vec &observation) const

  Return the probability that the given observation came from this distribution.
- double **Probability** (const arma::vec &observation, const size_t component) const

  Return the probability that the given observation came from the given Gaussian component in this distribution.
- arma::vec **Random** () const

  Return a randomly generated observation according to the probability distribution defined by this object.
- void **Save** (const std::string &filename) const

*Save a **GMM** (p. 208) to an XML file.*

- const arma::vec & **Weights** () const

  Return a const reference to the a priori weights of each Gaussian.

- arma::vec & **Weights** ()

  Return a reference to the a priori weights of each Gaussian.

Private Member Functions

- double **LogLikelihood** (const arma::mat &dataPoints, const std::vector< arma::vec > &**means**, const std::vector< arma::mat > &covars, const arma::vec &**weights**) const

  This function computes the loglikelihood of the given model.

Private Attributes

- std::vector< arma::mat > **covariances**

  Vector of covariances; one for each Gaussian.

- size_t **dimensionality**

  The dimensionality of the model.

- FittingType & **fitter**

  Reference to the fitting object we should use.

- size_t **gaussians**

  The number of Gaussians in the model.

- FittingType **localFitter**

  Locally-stored fitting object; in case the user did not pass one.

- std::vector< arma::vec > **means**

  Vector of means; one for each Gaussian.

- arma::vec **weights**

  Vector of a priori weights for each Gaussian.

21.21.1 Detailed Description

template<typename FittingType = EMFit<>>class mlpack::gmm::GMM< FittingType >

A Gaussian Mixture Model (**GMM** (p. 208)).

This class uses maximum likelihood loss functions to estimate the parameters of the **GMM** (p. 208) on a given dataset via the given fitting mechanism, defined by the FittingType template parameter. The **GMM** (p. 208) can be trained using normal data, or data with probabilities of being from this **GMM** (p. 208) (see **GMM::Estimate()** (p. 214) for more information).

The FittingType template class must provide a way for the **GMM** (p. 208) to train on data. It must provide the following two functions:

```
void Estimate(const arma::mat& observations,
              std::vector<arma::vec>& means,
              std::vector<arma::mat>& covariances,
              arma::vec& weights);

void Estimate(const arma::mat& observations,
              const arma::vec& probabilities,
              std::vector<arma::vec>& means,
              std::vector<arma::mat>& covariances,
              arma::vec& weights);
```

These functions should produce a trained **GMM** (p. 208) from the given observations and probabilities. These may modify the size of the model (by increasing the size of the mean and covariance vectors as well as the weight vectors), but the method should expect that these vectors are already set to the size of the **GMM** (p. 208) as specified in the constructor.

For a sample implementation, see the **EMFit** (p. 202) class; this class uses the EM algorithm to train a **GMM** (p. 208), and is the default fitting type.

The **GMM** (p. 208), once trained, can be used to generate random points from the distribution and estimate the probability of points being from the distribution. The parameters of the **GMM** (p. 208) can be obtained through the accessors and mutators.

Example use:

```
// Set up a mixture of 5 gaussians in a 4-dimensional space (uses the default
// EM fitting mechanism).
GMM<> g(5, 4);

// Train the GMM given the data observations.
g.Estimate(data);

// Get the probability of 'observation' being observed from this GMM.
double probability = g.Probability(observation);

// Get a random observation from the GMM.
arma::vec observation = g.Random();
```

Definition at line 89 of file gmm.hpp.

21.21.2 Constructor & Destructor Documentation

21.21.2.1 template<typename FittingType = EMFit<>> mlpack::gmm::GMM< FittingType >::GMM ( ) [inline]

Create an empty Gaussian Mixture Model, with zero gaussians.

Definition at line 107 of file gmm.hpp.

References mlpack::Log::Debug.

21.21.2.2 template<typename FittingType = EMFit<>> mlpack::gmm::GMM< FittingType >::GMM (const size_t *gaussians*, const size_t *dimensionality*) [inline]

Create a **GMM** (p. 208) with the given number of Gaussians, each of which have the specified dimensionality.

Parameters

gaussians	Number of Gaussians in this **GMM** (p. 208).
dimensionality	Dimensionality of each Gaussian.

Definition at line 127 of file gmm.hpp.

21.21.2.3 template<typename FittingType = EMFit<>> mlpack::gmm::GMM< FittingType >::GMM (const size_t *gaussians*, const size_t *dimensionality*, FittingType & *fitter*) [inline]

Create a **GMM** (p. 208) with the given number of Gaussians, each of which have the specified dimensionality.

Also, pass in an initialized FittingType class; this is useful in cases where the FittingType class needs to store some state.

Parameters

gaussians	Number of Gaussians in this **GMM** (p. 208).
dimensionality	Dimensionality of each Gaussian.
fitter	Initialized fitting mechanism.

Definition at line 146 of file gmm.hpp.

21.21.2.4 template<typename FittingType = EMFit<>> mlpack::gmm::GMM< FittingType >::GMM (const std::vector< arma::vec > & *means*, const std::vector< arma::mat > & *covariances*, const arma::vec & *weights*) [inline]

Create a **GMM** (p. 208) with the given means, covariances, and weights.

Parameters

means	Means of the model.
covariances	Covariances of the model.
weights	Weights of the model.

Definition at line 163 of file gmm.hpp.

21.21.2.5 template<typename FittingType = EMFit<>> mlpack::gmm::GMM< FittingType >::GMM (const std::vector< arma::vec > & *means*, const std::vector< arma::mat > & *covariances*, const arma::vec & *weights*, FittingType & *fitter*) [inline]

Create a **GMM** (p. 208) with the given means, covariances, and weights, and use the given initialized FittingType class.

This is useful in cases where the FittingType class needs to store some state.

Parameters

means	Means of the model.
covariances	Covariances of the model.
weights	Weights of the model.

Definition at line 183 of file gmm.hpp.

21.21.2.6 template<typename FittingType = EMFit<>> template<typename OtherFittingType > mlpack::gmm::GMM< FittingType >::GMM (const GMM< OtherFittingType > & *other*)

Copy constructor for GMMs which use different fitting types.

21.21.2.7 template<typename FittingType = EMFit<>> mlpack::gmm::GMM< FittingType >::GMM (const GMM< FittingType > & *other*)

Copy constructor for GMMs using the same fitting type.

This also copies the fitter.

21.21.3 Member Function Documentation

21.21.3.1 template<typename FittingType = EMFit<>> void mlpack::gmm::GMM< FittingType >::Classify (const arma::mat & *observations*, arma::Col< size_t > & *labels*) const

Classify the given observations as being from an individual component in this **GMM** (p. 208).

The resultant classifications are stored in the 'labels' object, and each label will be between 0 and (**Gaussians()** (p. 215) - 1). Supposing that a point was classified with label 2, and that our **GMM** (p. 208) object was called 'gmm', one could access the relevant Gaussian distribution as follows:

```
arma::vec mean = gmm.Means()[2];
arma::mat covariance = gmm.Covariances()[2];
double priorWeight = gmm.Weights()[2];
```

Parameters

observations	List of observations to classify.
labels	Object which will be filled with labels.

21.21.3.2 template<typename FittingType = EMFit<>> const std::vector<arma::mat>& mlpack::gmm::GMM< FittingType >::Covariances ( ) const [inline]

Return a const reference to the vector of covariance matrices (sigma).

Definition at line 251 of file gmm.hpp.

References mlpack::gmm::GMM< FittingType >::covariances.

21.21.3.3 template<typename FittingType = EMFit<>> std::vector<arma::mat>& mlpack::gmm::GMM< FittingType >::Covariances ( ) [inline]

Return a reference to the vector of covariance matrices (sigma).

Definition at line 253 of file gmm.hpp.

References mlpack::gmm::GMM< FittingType >::covariances.

21.21.3.4 template<typename FittingType = EMFit<>> size_t mlpack::gmm::GMM< FittingType >::Dimensionality ( ) const [inline]

Return the dimensionality of the model.

Definition at line 240 of file gmm.hpp.

References mlpack::gmm::GMM< FittingType >::dimensionality.

21.21.3.5 template<typename FittingType = EMFit<>> size_t& mlpack::gmm::GMM< FittingType >::Dimensionality ( ) [inline]

Modify the dimensionality of the model.

Careful! You will have to update each mean and covariance matrix yourself.

Definition at line 243 of file gmm.hpp.

References mlpack::gmm::GMM< FittingType >::dimensionality.

21.21.3.6 template<typename FittingType = EMFit<>> double mlpack::gmm::GMM< FittingType >::Estimate (const arma::mat & *observations*, const size_t *trials* = 1, const bool *useExistingModel* = false)

Estimate the probability distribution directly from the given observations, using the given algorithm in the FittingType class to fit the data.

The fitting will be performed 'trials' times; from these trials, the model with the greatest log-likelihood will be selected. By default, only one trial is performed. The log-likelihood of the best fitting is returned.

Optionally, the existing model can be used as an initial model for the estimation by setting 'useExistingModel' to true. If the fitting procedure is deterministic after the initial position is given, then 'trials' should be set to 1.

Template Parameters

FittingType	The type of fitting method which should be used (EMFit<> is suggested).

Parameters

observations	Observations of the model.
trials	Number of trials to perform; the model in these trials with the greatest log-likelihood will be selected.
useExistingModel	If true, the existing model is used as an initial model for the estimation.

Returns

The log-likelihood of the best fit.

21.21.3.7 template<typename FittingType = EMFit<>> double mlpack::gmm::GMM< FittingType >::Estimate (const arma::mat & *observations*, const arma::vec & *probabilities*, const size_t *trials* = 1, const bool *useExistingModel* = false)

Estimate the probability distribution directly from the given observations, taking into account the probability of each observation actually being from this distribution, and using the given algorithm in the FittingType class to fit the data.

The fitting will be performed 'trials' times; from these trials, the model with the greatest log-likelihood will be selected. By default, only one trial is performed. The log-likelihood of the best fitting is returned.

Optionally, the existing model can be used as an initial model for the estimation by setting 'useExistingModel' to true. If the fitting procedure is deterministic after the initial position is given, then 'trials' should be set to 1.

Parameters

observations	Observations of the model.
probabilities	Probability of each observation being from this distribution.
trials	Number of trials to perform; the model in these trials with the greatest log-likelihood will be selected.
useExistingModel	If true, the existing model is used as an initial model for the estimation.

Returns

The log-likelihood of the best fit.

21.21.3.8 template<typename FittingType = EMFit<>> const FittingType& mlpack::gmm::GMM< FittingType >::Fitter ( ) const [inline]

Return a const reference to the fitting type.

Definition at line 261 of file gmm.hpp.

References mlpack::gmm::GMM< FittingType >::fitter.

21.21.3.9 template<typename FittingType = EMFit<>> FittingType& mlpack::gmm::GMM< FittingType >::Fitter ( ) [inline]

Return a reference to the fitting type.

Definition at line 263 of file gmm.hpp.

References mlpack::gmm::GMM< FittingType >::fitter.

21.21.3.10 template<typename FittingType = EMFit<>> size_t mlpack::gmm::GMM< FittingType >::Gaussians ( ) const [inline]

Return the number of gaussians in the model.

Definition at line 234 of file gmm.hpp.

References mlpack::gmm::GMM< FittingType >::gaussians.

21.21.3.11 template<typename FittingType = EMFit<>> size_t& mlpack::gmm::GMM< FittingType >::Gaussians ( ) [inline]

Modify the number of gaussians in the model.

Careful! You will have to resize the means, covariances, and weights yourself.

Definition at line 237 of file gmm.hpp.

References mlpack::gmm::GMM< FittingType >::gaussians.

21.21.3.12 template<typename FittingType = EMFit<>> void mlpack::gmm::GMM< FittingType >::Load (const std::string & *filename*)

Load a **GMM** (p. 208) from an XML file.

The format of the XML file should be the same as is generated by the **Save()** (p. 217) method.

Parameters

filename	Name of XML file containing model to be loaded.

21.21.3.13 template<typename FittingType = EMFit<>> double mlpack::gmm::GMM< FittingType >::LogLikelihood (const arma::mat & *dataPoints,* const std::vector< arma::vec > & *means,* const std::vector< arma::mat > & *covars,* const arma::vec & *weights*) const [private]

This function computes the loglikelihood of the given model.

This function is used by **GMM::Estimate()** (p. 214).

Parameters

dataPoints	Observations to calculate the likelihood for.
means	Means of the given mixture model.
covars	Covariances of the given mixture model.
weights	Weights of the given mixture model.

21.21.3.14 template<typename FittingType = EMFit<>> const std::vector<arma::vec>& mlpack::gmm::GMM< FittingType >::Means ( ) const [inline]

Return a const reference to the vector of means (mu).

Definition at line 246 of file gmm.hpp.

References mlpack::gmm::GMM< FittingType >::means.

21.21.3.15 template<typename FittingType = EMFit<>> std::vector<arma::vec>& mlpack::gmm::GMM< FittingType >::Means ( ) [inline]

Return a reference to the vector of means (mu).

Definition at line 248 of file gmm.hpp.

References mlpack::gmm::GMM< FittingType >::means.

21.21.3.16 template<typename FittingType = EMFit<>> template<typename OtherFittingType > GMM& mlpack::gmm::GMM< FittingType >::operator= (const GMM< OtherFittingType > & *other*)

Copy operator for GMMs which use different fitting types.

21.21.3.17 template<typename FittingType = EMFit<>> GMM& mlpack::gmm::GMM< FittingType >::operator= (const GMM< FittingType > & *other*)

Copy operator for GMMs which use the same fitting type.

This also copies the fitter.

21.21.3.18 template<typename FittingType = EMFit<>> double mlpack::gmm::GMM< FittingType >::Probability (const arma::vec & *observation*) const

Return the probability that the given observation came from this distribution.

Parameters

observation	Observation to evaluate the probability of.

21.21.3.19 template<typename FittingType = EMFit<>> double mlpack::gmm::GMM< FittingType >::Probability (const arma::vec & *observation,* const size_t *component*) const

Return the probability that the given observation came from the given Gaussian component in this distribution.

Parameters

observation	Observation to evaluate the probability of.
component	Index of the component of the **GMM** (p. 208) to be considered.

21.21.3.20 template<typename FittingType = EMFit<>> arma::vec mlpack::gmm::GMM< FittingType >::Random ( ) const

Return a randomly generated observation according to the probability distribution defined by this object.

Returns

Random observation from this **GMM** (p. 208).

21.21.3.21 template<typename FittingType = EMFit<>> void mlpack::gmm::GMM< FittingType >::Save (const std::string & *filename*) const

Save a **GMM** (p. 208) to an XML file.

Parameters

filename	Name of XML file to write to.

21.21.3.22 template<typename FittingType = EMFit<>> const arma::vec& mlpack::gmm::GMM< FittingType >::Weights ( ) const [inline]

Return a const reference to the a priori weights of each Gaussian.

Definition at line 256 of file gmm.hpp.

References mlpack::gmm::GMM< FittingType >::weights.

21.21.3.23 template<typename FittingType = EMFit<>> arma::vec& mlpack::gmm::GMM< FittingType >::Weights ( ) [inline]

Return a reference to the a priori weights of each Gaussian.

Definition at line 258 of file gmm.hpp.

References mlpack::gmm::GMM< FittingType >::weights.

21.21.4 Member Data Documentation

21.21.4.1 template<typename FittingType = EMFit<>> std::vector<arma::mat> mlpack::gmm::GMM< FittingType >::covariances [private]

Vector of covariances; one for each Gaussian.

Definition at line 99 of file gmm.hpp.

Referenced by mlpack::gmm::GMM< FittingType >::Covariances().

21.21.4.2 template<typename FittingType = EMFit<>> size_t mlpack::gmm::GMM< FittingType >::dimensionality [private]

The dimensionality of the model.

Definition at line 95 of file gmm.hpp.

Referenced by mlpack::gmm::GMM< FittingType >::Dimensionality().

21.21.4.3 template<typename FittingType = EMFit<>> FittingType& mlpack::gmm::GMM< FittingType >::fitter [private]

Reference to the fitting object we should use.

Definition at line 384 of file gmm.hpp.

Referenced by mlpack::gmm::GMM< FittingType >::Fitter().

21.21.4.4 template<typename FittingType = EMFit<>> size_t mlpack::gmm::GMM< FittingType >::gaussians [private]

The number of Gaussians in the model.

Definition at line 93 of file gmm.hpp.

Referenced by mlpack::gmm::GMM< FittingType >::Gaussians().

21.21.4.5 template<typename FittingType = EMFit<>> FittingType mlpack::gmm::GMM< FittingType >::localFitter [private]

Locally-stored fitting object; in case the user did not pass one.

Definition at line 381 of file gmm.hpp.

21.21.4.6 template<typename FittingType = EMFit<>> std::vector<arma::vec> mlpack::gmm::GMM< FittingType >::means [private]

Vector of means; one for each Gaussian.

Definition at line 97 of file gmm.hpp.

Referenced by mlpack::gmm::GMM< FittingType >::Means().

21.21.4.7 template<typename FittingType = EMFit<>> arma::vec mlpack::gmm::GMM< FittingType >::weights [private]

Vector of a priori weights for each Gaussian.

Definition at line 101 of file gmm.hpp.

Referenced by mlpack::gmm::GMM< FittingType >::Weights().

The documentation for this class was generated from the following file:

- src/mlpack/methods/gmm/**gmm.hpp**

21.22 mlpack::gmm::NoConstraint Class Reference

This class enforces no constraint on the covariance matrix.

Static Public Member Functions

- static void **ApplyConstraint** (const arma::mat &)

  Do nothing, and do not modify the covariance matrix.

21.22.1 Detailed Description

This class enforces no constraint on the covariance matrix.

It's faster this way, although depending on your situation you may end up with a non-invertible covariance matrix.

Definition at line 35 of file no_constraint.hpp.

21.22.2 Member Function Documentation

21.22.2.1 static void mlpack::gmm::NoConstraint::ApplyConstraint (const arma::mat &) [inline], [static]

Do nothing, and do not modify the covariance matrix.

Definition at line 39 of file no_constraint.hpp.

The documentation for this class was generated from the following file:

- src/mlpack/methods/gmm/**no_constraint.hpp**

21.23 mlpack::gmm::PositiveDefiniteConstraint Class Reference

Given a covariance matrix, force the matrix to be positive definite.

Static Public Member Functions

- static void **ApplyConstraint** (arma::mat &covariance)

  Apply the positive definiteness constraint to the given covariance matrix.

21.23.1 Detailed Description

Given a covariance matrix, force the matrix to be positive definite.

Definition at line 31 of file positive_definite_constraint.hpp.

21.23.2 Member Function Documentation

21.23.2.1 static void mlpack::gmm::PositiveDefiniteConstraint::ApplyConstraint (arma::mat & *covariance*) [inline], [static]

Apply the positive definiteness constraint to the given covariance matrix.

Parameters

covariance	Covariance matrix.

Definition at line 39 of file positive_definite_constraint.hpp.

References mlpack::Log::Debug.

The documentation for this class was generated from the following file:

- src/mlpack/methods/gmm/**positive_definite_constraint.hpp**

21.24 mlpack::hmm::HMM< Distribution > Class Template Reference

A class that represents a Hidden Markov Model with an arbitrary type of emission distribution.

Public Member Functions

- **HMM** (const size_t states, const Distribution emissions, const double **tolerance**=1e-5)

  Create the Hidden Markov Model with the given number of hidden states and the given default distribution for emissions.
- **HMM** (const arma::mat &**transition**, const std::vector< Distribution > &**emission**, const double **tolerance**=1e-5)

  Create the Hidden Markov Model with the given transition matrix and the given emission distributions.
- size_t **Dimensionality** () const

  Get the dimensionality of observations.
- size_t & **Dimensionality** ()

  Set the dimensionality of observations.
- const std::vector< Distribution > & **Emission** () const

  Return the emission distributions.
- std::vector< Distribution > & **Emission** ()

  Return a modifiable emission probability matrix reference.
- double **Estimate** (const arma::mat &dataSeq, arma::mat &stateProb, arma::mat &forwardProb, arma::mat &backwardProb, arma::vec &scales) const

  Estimate the probabilities of each hidden state at each time step for each given data observation, using the Forward--Backward algorithm.
- double **Estimate** (const arma::mat &dataSeq, arma::mat &stateProb) const

  Estimate the probabilities of each hidden state at each time step of each given data observation, using the Forward--Backward algorithm.
- void **Generate** (const size_t length, arma::mat &dataSequence, arma::Col< size_t > &stateSequence, const size_t startState=0) const

  Generate a random data sequence of the given length.
- double **LogLikelihood** (const arma::mat &dataSeq) const

  Compute the log-likelihood of the given data sequence.
- double **Predict** (const arma::mat &dataSeq, arma::Col< size_t > &stateSeq) const

  Compute the most probable hidden state sequence for the given data sequence, using the Viterbi algorithm, returning the log-likelihood of the most likely state sequence.
- double **Tolerance** () const

  Get the tolerance of the Baum-Welch algorithm.
- double & **Tolerance** ()

  Modify the tolerance of the Baum-Welch algorithm.
- void **Train** (const std::vector< arma::mat > &dataSeq)

Train the model using the Baum-Welch algorithm, with only the given unlabeled observations.

- void **Train** (const std::vector< arma::mat > &dataSeq, const std::vector< arma::Col< size_t > > &stateSeq)

  Train the model using the given labeled observations; the transition and emission matrices are directly estimated.
- const arma::mat & **Transition** () const

  Return the transition matrix.
- arma::mat & **Transition** ()

  Return a modifiable transition matrix reference.

Private Member Functions

- void **Backward** (const arma::mat &dataSeq, const arma::vec &scales, arma::mat &backwardProb) const

  The Backward algorithm (part of the Forward-Backward algorithm).
- void **Forward** (const arma::mat &dataSeq, arma::vec &scales, arma::mat &forwardProb) const

  The Forward algorithm (part of the Forward-Backward algorithm).

Private Attributes

- size_t **dimensionality**

  Dimensionality of observations.
- std::vector< Distribution > **emission**

  Set of emission probability distributions; one for each state.
- double **tolerance**

  Tolerance of Baum-Welch algorithm.
- arma::mat **transition**

  Transition probability matrix.

21.24.1 Detailed Description

template<typename Distribution = distribution::DiscreteDistribution>class mlpack::hmm::HMM< Distribution >

A class that represents a Hidden Markov Model with an arbitrary type of emission distribution.

This **HMM** (p. 221) class supports training (supervised and unsupervised), prediction of state sequences via the Viterbi algorithm, estimation of state probabilities, generation of random sequences, and calculation of the log-likelihood of a given sequence.

The template parameter, Distribution, specifies the distribution which the emissions follow. The class should implement the following functions:

```
class Distribution
{
 public:
  // The type of observation used by this distribution.
  typedef something DataType;

  // Return the probability of the given observation.
  double Probability(const DataType& observation) const;

  // Estimate the distribution based on the given observations.
  void Estimate(const std::vector<DataType>& observations);

  // Estimate the distribution based on the given observations, given also
  // the probability of each observation coming from this distribution.
  void Estimate(const std::vector<DataType>& observations,
                const std::vector<double>& probabilities);
};
```

See the **mlpack::distribution::DiscreteDistribution** (p. 157) class for an example. One would use the DiscreteDistribution class when the observations are non-negative integers. Other distributions could be Gaussians, a mixture of Gaussians (GMM), or any other probability distribution implementing the four Distribution functions.

Usage of the **HMM** (p. 221) class generally involves either training an **HMM** (p. 221) or loading an already-known **HMM** (p. 221) and taking probability measurements of sequences. Example code for supervised training of a Gaussian **HMM** (p. 221) (that is, where the emission output distribution is a single Gaussian for each hidden state) is given below.

```
extern arma::mat observations; // Each column is an observation.
extern arma::Col<size_t> states; // Hidden states for each observation.
// Create an untrained HMM with 5 hidden states and default (N(0, 1))
// Gaussian distributions with the dimensionality of the dataset.
HMM<GaussianDistribution> hmm(5, GaussianDistribution(observations.n_rows));

// Train the HMM (the labels could be omitted to perform unsupervised
// training).
hmm.Train(observations, states);
```

Once initialized, the **HMM** (p. 221) can evaluate the probability of a certain sequence (with **LogLikelihood()** (p. 226)), predict the most likely sequence of hidden states (with **Predict()** (p. 226)), generate a sequence (with **Generate()** (p. 226)), or estimate the probabilities of each state for a sequence of observations (with **Estimate()** (p. 225)).

Template Parameters

Distribution	Type of emission distribution for this **HMM** (p. 221).

Definition at line 93 of file hmm.hpp.

21.24.2 Constructor & Destructor Documentation

21.24.2.1 template<typename Distribution = distribution::DiscreteDistribution> mlpack::hmm::HMM< Distribution >::HMM (const size_t *states*, const Distribution *emissions*, const double *tolerance* = 1e-5)

Create the Hidden Markov Model with the given number of hidden states and the given default distribution for emissions.

The dimensionality of the observations is taken from the emissions variable, so it is important that the given default emission distribution is set with the correct dimensionality. Alternately, set the dimensionality with **Dimensionality()** (p. 224). Optionally, the tolerance for convergence of the Baum-Welch algorithm can be set.

Parameters

states	Number of states.
emissions	Default distribution for emissions.
tolerance	Tolerance for convergence of training algorithm (Baum-Welch).

21.24.2.2 template<typename Distribution = distribution::DiscreteDistribution> mlpack::hmm::HMM< Distribution >::HMM (const arma::mat & *transition*, const std::vector< Distribution > & *emission*, const double *tolerance* = 1e-5)

Create the Hidden Markov Model with the given transition matrix and the given emission distributions.

The dimensionality of the observations of the **HMM** (p. 221) are taken from the given emission distributions. Alternately, the dimensionality can be set with **Dimensionality()** (p. 224).

The transition matrix should be such that T(i, j) is the probability of transition to state i from state j. The columns of the matrix should sum to 1.

The emission matrix should be such that E(i, j) is the probability of emission i while in state j. The columns of the matrix should sum to 1.

Optionally, the tolerance for convergence of the Baum-Welch algorithm can be set.

Parameters

transition	Transition matrix.
emission	Emission distributions.
tolerance	Tolerance for convergence of training algorithm (Baum-Welch).

21.24.3 Member Function Documentation

21.24.3.1 template<typename Distribution = distribution::DiscreteDistribution> void mlpack::hmm::HMM< Distribution >::Backward (const arma::mat & *dataSeq*, const arma::vec & *scales*, arma::mat & *backwardProb*) const [private]

The Backward algorithm (part of the Forward-Backward algorithm).

Computes backward probabilities for each state for each observation in the given data sequence, using the scaling factors found (presumably) by **Forward()** (p. 225). The returned matrix has rows equal to the number of hidden states and columns equal to the number of observations.

Parameters

dataSeq	Data sequence to compute probabilities for.
scales	Vector of scaling factors.
backwardProb	Matrix in which backward probabilities will be saved.

21.24.3.2 template<typename Distribution = distribution::DiscreteDistribution> size_t mlpack::hmm::HMM< Distribution >::Dimensionality ( ) const [inline]

Get the dimensionality of observations.

Definition at line 279 of file hmm.hpp.

References mlpack::hmm::HMM< Distribution >::dimensionality.

21.24.3.3 template<typename Distribution = distribution::DiscreteDistribution> size_t& mlpack::hmm::HMM< Distribution >::Dimensionality ( ) [inline]

Set the dimensionality of observations.

Definition at line 281 of file hmm.hpp.

References mlpack::hmm::HMM< Distribution >::dimensionality.

21.24.3.4 template<typename Distribution = distribution::DiscreteDistribution> const std::vector<Distribution>& mlpack::hmm::HMM< Distribution >::Emission ( ) const [inline]

Return the emission distributions.

Definition at line 274 of file hmm.hpp.

References mlpack::hmm::HMM< Distribution >::emission.

21.24.3.5 template<typename Distribution = distribution::DiscreteDistribution> std::vector<Distribution>& mlpack::hmm::HMM< Distribution >::Emission ( ) [inline]

Return a modifiable emission probability matrix reference.

Definition at line 276 of file hmm.hpp.

References mlpack::hmm::HMM< Distribution >::emission.

21.24.3.6 template<typename Distribution = distribution::DiscreteDistribution> double mlpack::hmm::HMM< Distribution >::Estimate (const arma::mat & *dataSeq,* arma::mat & *stateProb,* arma::mat & *forwardProb,* arma::mat & *backwardProb,* arma::vec & *scales*) const

Estimate the probabilities of each hidden state at each time step for each given data observation, using the Forward--Backward algorithm.

Each matrix which is returned has columns equal to the number of data observations, and rows equal to the number of hidden states in the model. The log-likelihood of the most probable sequence is returned.

Parameters

dataSeq	Sequence of observations.
stateProb	Matrix in which the probabilities of each state at each time interval will be stored.
forwardProb	Matrix in which the forward probabilities of each state at each time interval will be stored.
backwardProb	Matrix in which the backward probabilities of each state at each time interval will be stored.
scales	Vector in which the scaling factors at each time interval will be stored.

Returns

Log-likelihood of most likely state sequence.

21.24.3.7 template<typename Distribution = distribution::DiscreteDistribution> double mlpack::hmm::HMM< Distribution >::Estimate (const arma::mat & *dataSeq,* arma::mat & *stateProb*) const

Estimate the probabilities of each hidden state at each time step of each given data observation, using the Forward--Backward algorithm.

The returned matrix of state probabilities has columns equal to the number of data observations, and rows equal to the number of hidden states in the model. The log-likelihood of the most probable sequence is returned.

Parameters

dataSeq	Sequence of observations.
stateProb	Probabilities of each state at each time interval.

Returns

Log-likelihood of most likely state sequence.

21.24.3.8 template<typename Distribution = distribution::DiscreteDistribution> void mlpack::hmm::HMM< Distribution >::Forward (const arma::mat & *dataSeq,* arma::vec & *scales,* arma::mat & *forwardProb*) const [private]

The Forward algorithm (part of the Forward-Backward algorithm).

Computes forward probabilities for each state for each observation in the given data sequence. The returned matrix has rows equal to the number of hidden states and columns equal to the number of observations.

Parameters

dataSeq	Data sequence to compute probabilities for.
scales	Vector in which scaling factors will be saved.
forwardProb	Matrix in which forward probabilities will be saved.

21.24.3.9 template<typename Distribution = distribution::DiscreteDistribution> void mlpack::hmm::HMM< Distribution >::Generate (const size_t *length*, arma::mat & *dataSequence*, arma::Col< size_t > & *stateSequence*, const size_t *startState* = 0) const

Generate a random data sequence of the given length.

The data sequence is stored in the dataSequence parameter, and the state sequence is stored in the stateSequence parameter. Each column of dataSequence represents a random observation.

Parameters

length	Length of random sequence to generate.
dataSequence	Vector to store data in.
stateSequence	Vector to store states in.
startState	Hidden state to start sequence in (default 0).

21.24.3.10 template<typename Distribution = distribution::DiscreteDistribution> double mlpack::hmm::HMM< Distribution >::LogLikelihood (const arma::mat & *dataSeq*) const

Compute the log-likelihood of the given data sequence.

Parameters

dataSeq	Data sequence to evaluate the likelihood of.

Returns

Log-likelihood of the given sequence.

21.24.3.11 template<typename Distribution = distribution::DiscreteDistribution> double mlpack::hmm::HMM< Distribution >::Predict (const arma::mat & *dataSeq*, arma::Col< size_t > & *stateSeq*) const

Compute the most probable hidden state sequence for the given data sequence, using the Viterbi algorithm, returning the log-likelihood of the most likely state sequence.

Parameters

dataSeq	Sequence of observations.
stateSeq	Vector in which the most probable state sequence will be stored.

Returns

Log-likelihood of most probable state sequence.

21.24.3.12 template<typename Distribution = distribution::DiscreteDistribution> double mlpack::hmm::HMM< Distribution >::Tolerance ( ) const [inline]

Get the tolerance of the Baum-Welch algorithm.

Definition at line 284 of file hmm.hpp.

References mlpack::hmm::HMM< Distribution >::tolerance.

21.24.3.13 template<typename Distribution = distribution::DiscreteDistribution> double& mlpack::hmm::HMM< Distribution >::Tolerance ( ) [inline]

Modify the tolerance of the Baum-Welch algorithm.

Definition at line 286 of file hmm.hpp.

References mlpack::hmm::HMM< Distribution >::tolerance.

21.24.3.14 template<typename Distribution = distribution::DiscreteDistribution> void mlpack::hmm::HMM< Distribution >::Train (const std::vector< arma::mat > & *dataSeq*)

Train the model using the Baum-Welch algorithm, with only the given unlabeled observations.

Instead of giving a guess transition and emission matrix here, do that in the constructor. Each matrix in the vector of data sequences holds an individual data sequence; each point in each individual data sequence should be a column in the matrix. The number of rows in each matrix should be equal to the dimensionality of the **HMM** (p. 221) (which is set in the constructor).

It is preferable to use the other overload of **Train()** (p. 227), with labeled data. That will produce much better results. However, if labeled data is unavailable, this will work. In addition, it is possible to use **Train()** (p. 227) with labeled data first, and then continue to train the model using this overload of **Train()** (p. 227) with unlabeled data.

The tolerance of the Baum-Welch algorithm can be set either in the constructor or with the **Tolerance()** (p. 227) method. When the change in log-likelihood of the model between iterations is less than the tolerance, the Baum-Welch algorithm terminates.

Note

Train() (p. 227) can be called multiple times with different sequences; each time it is called, it uses the current parameters of the **HMM** (p. 221) as a starting point for training.

Parameters

dataSeq	Vector of observation sequences.

21.24.3.15 template<typename Distribution = distribution::DiscreteDistribution> void mlpack::hmm::HMM< Distribution >::Train (const std::vector< arma::mat > & *dataSeq*, const std::vector< arma::Col< size_t > > & *stateSeq*)

Train the model using the given labeled observations; the transition and emission matrices are directly estimated.

Each matrix in the vector of data sequences corresponds to a vector in the vector of state sequences. Each point in each individual data sequence should be a column in the matrix, and its state should be the corresponding element in the state sequence vector. For instance, dataSeq[0].col(3) corresponds to the fourth observation in the first data sequence, and its state is stateSeq[0][3]. The number of rows in each matrix should be equal to the dimensionality of the **HMM** (p. 221) (which is set in the constructor).

Note

Train() (p. 227) can be called multiple times with different sequences; each time it is called, it uses the current parameters of the **HMM** (p. 221) as a starting point for training.

Parameters

dataSeq	Vector of observation sequences.
stateSeq	Vector of state sequences, corresponding to each observation.

21.24.3.16 template<typename Distribution = distribution::DiscreteDistribution> const arma::mat& mlpack::hmm::HMM< Distribution >::Transition ( ) const [inline]

Return the transition matrix.

Definition at line 269 of file hmm.hpp.

References mlpack::hmm::HMM< Distribution >::transition.

21.24.3.17 template<typename Distribution = distribution::DiscreteDistribution> arma::mat& mlpack::hmm::HMM< Distribution >::Transition ( ) [inline]

Return a modifiable transition matrix reference.

Definition at line 271 of file hmm.hpp.

References mlpack::hmm::HMM< Distribution >::transition.

21.24.4 Member Data Documentation

21.24.4.1 template<typename Distribution = distribution::DiscreteDistribution> size_t mlpack::hmm::HMM< Distribution >::dimensionality [private]

Dimensionality of observations.

Definition at line 327 of file hmm.hpp.

Referenced by mlpack::hmm::HMM< Distribution >::Dimensionality().

21.24.4.2 template<typename Distribution = distribution::DiscreteDistribution> std::vector<Distribution> mlpack::hmm::HMM< Distribution >::emission [private]

Set of emission probability distributions; one for each state.

Definition at line 324 of file hmm.hpp.

Referenced by mlpack::hmm::HMM< Distribution >::Emission().

21.24.4.3 template<typename Distribution = distribution::DiscreteDistribution> double mlpack::hmm::HMM< Distribution >::tolerance [private]

Tolerance of Baum-Welch algorithm.

Definition at line 330 of file hmm.hpp.

Referenced by mlpack::hmm::HMM< Distribution >::Tolerance().

21.24.4.4 template<typename Distribution = distribution::DiscreteDistribution> arma::mat mlpack::hmm::HMM< Distribution >::transition [private]

Transition probability matrix.

Definition at line 321 of file hmm.hpp.

Referenced by mlpack::hmm::HMM< Distribution >::Transition().

The documentation for this class was generated from the following file:

- src/mlpack/methods/hmm/**hmm.hpp**

21.25 mlpack::kernel::CosineDistance Class Reference

The cosine distance (or cosine similarity).

Static Public Member Functions

- template<typename VecType >
  static double **Evaluate** (const VecType &a, const VecType &b)

  Computes the cosine distance between two points.

21.25.1 Detailed Description

The cosine distance (or cosine similarity).

It is defined by

$$d(a,b) = \frac{a^T b}{||a||||b||}$$

and this class assumes the standard L2 inner product.

Definition at line 40 of file cosine_distance.hpp.

21.25.2 Member Function Documentation

21.25.2.1 template<typename VecType > static double mlpack::kernel::CosineDistance::Evaluate (const VecType & *a,* const VecType & *b*) [static]

Computes the cosine distance between two points.

Parameters

a	First vector.
b	Second vector.

Returns

d(a, b).

The documentation for this class was generated from the following file:

- src/mlpack/core/kernels/**cosine_distance.hpp**

21.26 mlpack::kernel::EpanechnikovKernel Class Reference

The Epanechnikov kernel, defined as.

Public Member Functions

- **EpanechnikovKernel** (const double **bandwidth**=1.0)

  Instantiate the Epanechnikov kernel with the given bandwidth (default 1.0).
- template<typename VecType >
  double **ConvolutionIntegral** (const VecType &a, const VecType &b)

  Obtains the convolution integral [integral of K(||x-a||) K(||b-x||) dx] for the two vectors.
- template<typename Vec1Type , typename Vec2Type >
  double **Evaluate** (const Vec1Type &a, const Vec2Type &b) const

  Evaluate the Epanechnikov kernel on the given two inputs.
- double **Evaluate** (const double distance) const

  Evaluate the Epanechnikov kernel given that the distance between the two input points is known.
- double **Normalizer** (const size_t dimension)

  Compute the normalizer of this Epanechnikov kernel for the given dimension.

Private Attributes

- double **bandwidth**

  Bandwidth of the kernel.
- double **inverseBandwidthSquared**

  Cached value of the inverse bandwidth squared (to speed up computation).

21.26.1 Detailed Description

The Epanechnikov kernel, defined as.

$$K(x,y) = \max\{0, 1 - ||x-y||_2^2 / b^2\}$$

where b is the bandwidth the of the kernel (defaults to 1.0).

Definition at line 39 of file epanechnikov_kernel.hpp.

21.26.2 Constructor & Destructor Documentation

21.26.2.1 mlpack::kernel::EpanechnikovKernel::EpanechnikovKernel (const double *bandwidth* = `1.0`) [inline]

Instantiate the Epanechnikov kernel with the given bandwidth (default 1.0).

Parameters

bandwidth	Bandwidth of the kernel.

Definition at line 47 of file epanechnikov_kernel.hpp.

21.26.3 Member Function Documentation

21.26.3.1 template<typename VecType > double mlpack::kernel::EpanechnikovKernel::ConvolutionIntegral (const VecType & *a,* const VecType & *b*)

Obtains the convolution integral [integral of K(||x-a||) K(||b-x||) dx] for the two vectors.

Template Parameters

VecType	Type of vector (arma::vec, arma::spvec should be expected).

Parameters

a	First vector.
b	Second vector.

Returns

the convolution integral value.

21.26.3.2 template<typename Vec1Type , typename Vec2Type > double mlpack::kernel::EpanechnikovKernel::Evaluate (const Vec1Type & *a,* const Vec2Type & *b*) const

Evaluate the Epanechnikov kernel on the given two inputs.

Parameters

a	One input vector.
b	The other input vector.

21.26.3.3 double mlpack::kernel::EpanechnikovKernel::Evaluate (const double *distance*) const

Evaluate the Epanechnikov kernel given that the distance between the two input points is known.

21.26.3.4 double mlpack::kernel::EpanechnikovKernel::Normalizer (const size_t *dimension*)

Compute the normalizer of this Epanechnikov kernel for the given dimension.

Parameters

dimension	Dimension to calculate the normalizer for.

21.26.4 Member Data Documentation

21.26.4.1 double mlpack::kernel::EpanechnikovKernel::bandwidth [private]

Bandwidth of the kernel.

Definition at line 88 of file epanechnikov_kernel.hpp.

21.26.4.2 double mlpack::kernel::EpanechnikovKernel::inverseBandwidthSquared [private]

Cached value of the inverse bandwidth squared (to speed up computation).

Definition at line 90 of file epanechnikov_kernel.hpp.

The documentation for this class was generated from the following file:

- src/mlpack/core/kernels/**epanechnikov_kernel.hpp**

21.27 mlpack::kernel::ExampleKernel Class Reference

An example kernel function.

Public Member Functions

- **ExampleKernel** ()

  The default constructor, which takes no parameters.

Static Public Member Functions

- template<typename VecType >
  static double **ConvolutionIntegral** (const VecType &a, const VecType &b)

  Obtains the convolution integral [integral K(||x-a||)K(||b-x||)dx] for the two vectors.
- template<typename VecType >
  static double **Evaluate** (const VecType &a, const VecType &b)

  Evaluates the kernel function for two given vectors.
- static double **Normalizer** (size_t dimension)

  Obtains the normalizing volume for the kernel with dimension $dimension$.

21.27.1 Detailed Description

An example kernel function.

This is not a useful kernel, but it implements the two functions necessary to satisfy the Kernel policy (so that a class can be used whenever an MLPACK method calls for a `typename Kernel` template parameter.

All that is necessary is a constructor and an **`Evaluate()`** (p. 233) function. More methods could be added; for instance, one useful idea is a constructor which takes parameters for a kernel (for instance, the width of the Gaussian for a Gaussian kernel). However, MLPACK methods cannot count on these various constructors existing, which is why most methods allow passing an already-instantiated kernel object (and by default the method will construct the kernel with the default constructor). So, for instance,

```
GaussianKernel k(5.0);
KDE<GaussianKernel> kde(dataset, k);
```

will set up KDE using a Gaussian kernel with a width of 5.0, but

```
KDE<GaussianKernel> kde(dataset);
```

will create the kernel with the default constructor. It is important (but not strictly mandatory) that your default constructor still gives a working kernel.

Note

Not all kernels require state. For instance, the regular dot product needs no parameters. In that case, no local variables are necessary and **Evaluate()** (p. 233) can (and should) be declared static. However, for greater generalization, MLPACK methods expect all kernels to require state and hence must store instantiated kernel functions; this is why a default constructor is necessary.

Definition at line 94 of file example_kernel.hpp.

21.27.2 Constructor & Destructor Documentation

21.27.2.1 mlpack::kernel::ExampleKernel::ExampleKernel ( ) [inline]

The default constructor, which takes no parameters.

Because our simple example kernel has no internal parameters that need to be stored, the constructor does not need to do anything. For a more complex example, see the **GaussianKernel** (p. 235), which stores an internal parameter.

Definition at line 103 of file example_kernel.hpp.

21.27.3 Member Function Documentation

21.27.3.1 template<typename VecType > static double mlpack::kernel::ExampleKernel::ConvolutionIntegral (const VecType & *a,* const VecType & *b*) [inline], [static]

Obtains the convolution integral [integral K(||x-a||)K(||b-x||)dx] for the two vectors.

In this case, because our simple example kernel has no internal parameters, we can declare the function static. For a more complex example which cannot be declared static, see the **GaussianKernel** (p. 235), which stores an internal parameter.

Template Parameters

VecType	Type of vector (arma::vec, arma::spvec should be expected).

Parameters

a	First vector.
b	Second vector.

Returns

the convolution integral value.

Definition at line 132 of file example_kernel.hpp.

21.27.3.2 template<typename VecType > static double mlpack::kernel::ExampleKernel::Evaluate (const VecType & *a,* const VecType & *b*) [inline], [static]

Evaluates the kernel function for two given vectors.

In this case, because our simple example kernel has no internal parameters, we can declare the function static. For a more complex example which cannot be declared static, see the **GaussianKernel** (p. 235), which stores an internal parameter.

Template Parameters

VecType	Type of vector (arma::vec, arma::spvec should be expected).

Parameters

a	First vector.
b	Second vector.

Returns

K(a, b).

Definition at line 117 of file example_kernel.hpp.

21.27.3.3 static double mlpack::kernel::ExampleKernel::Normalizer (size_t *dimension*) [inline], [static]

Obtains the normalizing volume for the kernel with dimension $dimension$.

In this case, because our simple example kernel has no internal parameters, we can declare the function static. For a more complex example which cannot be declared static, see the **GaussianKernel** (p. 235), which stores an internal parameter.

Parameters

dimension	the dimension of the space.

Returns

the normalization constant.

Definition at line 145 of file example_kernel.hpp.

The documentation for this class was generated from the following file:

- src/mlpack/core/kernels/**example_kernel.hpp**

21.28 mlpack::kernel::GaussianKernel Class Reference

The standard Gaussian kernel.

Public Member Functions

- **GaussianKernel** ()

  Default constructor; sets bandwidth to 1.0.
- **GaussianKernel** (double **bandwidth**)

  Construct the Gaussian kernel with a custom bandwidth.
- double **Bandwidth** () const

  Get the bandwidth.
- void **Bandwidth** (const double **bandwidth**)

  Modify the bandwidth.

- template<typename VecType >
  double **ConvolutionIntegral** (const VecType &a, const VecType &b)

  Obtain a convolution integral of the Gaussian kernel.
- template<typename VecType >
  double **Evaluate** (const VecType &a, const VecType &b) const

  Evaluation of the Gaussian kernel.
- double **Evaluate** (double t) const

  Evaluation of the Gaussian kernel given the distance between two points.
- double **Gamma** () const

  Get the precalculated constant.
- double **Normalizer** (size_t dimension)

  Obtain the normalization constant of the Gaussian kernel.

Private Attributes

- double **bandwidth**

  Kernel bandwidth.
- double **gamma**

  Precalculated constant depending on the bandwidth; $\gamma = -\frac{1}{2\mu^2}$.

21.28.1 Detailed Description

The standard Gaussian kernel.

Given two vectors x, y, and a bandwidth μ (set in the constructor),

$$K(x,y) = \exp(-\frac{||x-y||^2}{2\mu^2}).$$

The implementation is all in the header file because it is so simple.

Definition at line 43 of file gaussian_kernel.hpp.

21.28.2 Constructor & Destructor Documentation

21.28.2.1 mlpack::kernel::GaussianKernel::GaussianKernel ( ) [inline]

Default constructor; sets bandwidth to 1.0.

Definition at line 49 of file gaussian_kernel.hpp.

21.28.2.2 mlpack::kernel::GaussianKernel::GaussianKernel (double *bandwidth*) [inline]

Construct the Gaussian kernel with a custom bandwidth.

Parameters

bandwidth	The bandwidth of the kernel (μ).

Definition at line 57 of file gaussian_kernel.hpp.

21.28.3 Member Function Documentation

21.28.3.1 double mlpack::kernel::GaussianKernel::Bandwidth ( ) const [inline]

Get the bandwidth.

Definition at line 120 of file gaussian_kernel.hpp.

References bandwidth.

21.28.3.2 void mlpack::kernel::GaussianKernel::Bandwidth (const double *bandwidth*) [inline]

Modify the bandwidth.

This takes an argument because we must update the precalculated constant (gamma).

Definition at line 124 of file gaussian_kernel.hpp.

References bandwidth, and gamma.

21.28.3.3 template<typename VecType > double mlpack::kernel::GaussianKernel::ConvolutionIntegral (const VecType & *a,* const VecType & *b*) [inline]

Obtain a convolution integral of the Gaussian kernel.

Parameters

a,first	vector
b,second	vector

Returns

the convolution integral

Definition at line 112 of file gaussian_kernel.hpp.

References Evaluate(), mlpack::metric::LMetric< Power, TakeRoot >::Evaluate(), and Normalizer().

21.28.3.4 template<typename VecType > double mlpack::kernel::GaussianKernel::Evaluate (const VecType & *a,* const VecType & *b*) const [inline]

Evaluation of the Gaussian kernel.

This could be generalized to use any distance metric, not the Euclidean distance, but for now, the Euclidean distance is used.

Template Parameters

VecType	Type of vector (likely arma::vec or arma::spvec).

Parameters

a	First vector.
b	Second vector.

Returns

K(a, b) using the bandwidth (μ) specified in the constructor.

Definition at line 74 of file gaussian_kernel.hpp.

References mlpack::metric::LMetric< Power, TakeRoot >::Evaluate(), and gamma.

Referenced by ConvolutionIntegral().

21.28.3.5 double mlpack::kernel::GaussianKernel::Evaluate (double *t*) const [inline]

Evaluation of the Gaussian kernel given the distance between two points.

Parameters

t	The distance between the two points the kernel is evaluated on.

Returns

K(t) using the bandwidth (μ) specified in the constructor.

Definition at line 87 of file gaussian_kernel.hpp.

References gamma.

21.28.3.6 double mlpack::kernel::GaussianKernel::Gamma ( ) const [inline]

Get the precalculated constant.

Definition at line 131 of file gaussian_kernel.hpp.

References gamma.

21.28.3.7 double mlpack::kernel::GaussianKernel::Normalizer (size_t *dimension*) [inline]

Obtain the normalization constant of the Gaussian kernel.

Parameters

dimension	

Returns

the normalization constant

Definition at line 99 of file gaussian_kernel.hpp.

References bandwidth, and M_PI.

Referenced by ConvolutionIntegral().

21.28.4 Member Data Documentation

21.28.4.1 double mlpack::kernel::GaussianKernel::bandwidth [private]

Kernel bandwidth.

Definition at line 135 of file gaussian_kernel.hpp.

Referenced by Bandwidth(), and Normalizer().

21.28.4.2 double mlpack::kernel::GaussianKernel::gamma [private]

Precalculated constant depending on the bandwidth; $\gamma = -\frac{1}{2\mu^2}$.

Definition at line 139 of file gaussian_kernel.hpp.

Referenced by Bandwidth(), Evaluate(), and Gamma().

The documentation for this class was generated from the following file:

- src/mlpack/core/kernels/**gaussian_kernel.hpp**

21.29 mlpack::kernel::HyperbolicTangentKernel Class Reference

Hyperbolic tangent kernel.

Public Member Functions

- **HyperbolicTangentKernel** ()

  This constructor sets the default scale to 1.0 and offset to 0.0.
- **HyperbolicTangentKernel** (double **scale**, double **offset**)

  Construct the hyperbolic tangent kernel with custom scale factor and offset.
- template<typename VecType >
  double **Evaluate** (const VecType &a, const VecType &b)

  Evaluate the hyperbolic tangent kernel.
- double **Offset** () const

  Get offset for the kernel.
- double & **Offset** ()

  Modify offset for the kernel.
- double **Scale** () const

  Get scale factor.
- double & **Scale** ()

  Modify scale factor.

Private Attributes

- double **offset**
- double **scale**

21.29.1 Detailed Description

Hyperbolic tangent kernel.

For any two vectors x, y and a given scale s and offset t

$$K(x,y) = \tanh(s < x,y > +t)$$

Definition at line 38 of file hyperbolic_tangent_kernel.hpp.

21.29.2 Constructor & Destructor Documentation

21.29.2.1 mlpack::kernel::HyperbolicTangentKernel::HyperbolicTangentKernel ( ) [inline]

This constructor sets the default scale to 1.0 and offset to 0.0.

Definition at line 44 of file hyperbolic_tangent_kernel.hpp.

21.29.2.2 mlpack::kernel::HyperbolicTangentKernel::HyperbolicTangentKernel (double *scale,* double *offset*) [inline]

Construct the hyperbolic tangent kernel with custom scale factor and offset.

Parameters

scale	Scaling factor for <x, y>.
offset	Kernel offset.

Definition at line 54 of file hyperbolic_tangent_kernel.hpp.

21.29.3 Member Function Documentation

21.29.3.1 template<typename VecType > double mlpack::kernel::HyperbolicTangentKernel::Evaluate (const VecType & *a,* const VecType & *b*) [inline]

Evaluate the hyperbolic tangent kernel.

This evaluation uses Armadillo's dot() function.

Template Parameters

VecType	Type of vector (should be arma::vec or arma::spvec).

Parameters

a	First vector.
b	Second vector.

Returns

K(a, b).

Definition at line 68 of file hyperbolic_tangent_kernel.hpp.

References offset, and scale.

21.29.3.2 double mlpack::kernel::HyperbolicTangentKernel::Offset ( ) const [inline]

Get offset for the kernel.

Definition at line 79 of file hyperbolic_tangent_kernel.hpp.

References offset.

21.29.3.3 double& mlpack::kernel::HyperbolicTangentKernel::Offset ( ) [inline]

Modify offset for the kernel.

Definition at line 81 of file hyperbolic_tangent_kernel.hpp.

References offset.

21.29.3.4 double mlpack::kernel::HyperbolicTangentKernel::Scale ( ) const [inline]

Get scale factor.

Definition at line 74 of file hyperbolic_tangent_kernel.hpp.

References scale.

21.29.3.5 double& mlpack::kernel::HyperbolicTangentKernel::Scale ( ) [inline]

Modify scale factor.

Definition at line 76 of file hyperbolic_tangent_kernel.hpp.

References scale.

21.29.4 Member Data Documentation

21.29.4.1 double mlpack::kernel::HyperbolicTangentKernel::offset [private]

Definition at line 85 of file hyperbolic_tangent_kernel.hpp.

Referenced by Evaluate(), and Offset().

21.29.4.2 double mlpack::kernel::HyperbolicTangentKernel::scale [private]

Definition at line 84 of file hyperbolic_tangent_kernel.hpp.

Referenced by Evaluate(), and Scale().

The documentation for this class was generated from the following file:

- src/mlpack/core/kernels/**hyperbolic_tangent_kernel.hpp**

21.30 mlpack::kernel::KernelTraits< KernelType > Class Template Reference

This is a template class that can provide information about various kernels.

Static Public Attributes

- static const bool **IsNormalized** = false

  If true, then the kernel is normalized: K(x, x) = K(y, y) = 1 for all x.

21.30.1 Detailed Description

template<typename KernelType>class mlpack::kernel::KernelTraits< KernelType >

This is a template class that can provide information about various kernels.

By default, this class will provide the weakest possible assumptions on kernels, and each kernel should override values as necessary. If a kernel doesn't need to override a value, then there's no need to write a **KernelTraits** (p. 241) specialization for that class.

Definition at line 37 of file kernel_traits.hpp.

21.30.2 Member Data Documentation

21.30.2.1 template<typename KernelType > const bool mlpack::kernel::KernelTraits< KernelType >::IsNormalized = false [static]

If true, then the kernel is normalized: K(x, x) = K(y, y) = 1 for all x.

Definition at line 43 of file kernel_traits.hpp.

The documentation for this class was generated from the following file:

- src/mlpack/core/kernels/**kernel_traits.hpp**

21.31 mlpack::kernel::KernelTraits< CosineDistance > Class Template Reference

Kernel traits for the cosine distance.

Static Public Attributes

- static const bool **IsNormalized** = true

  The cosine kernel is normalized: K(x, x) = 1 for all x.

21.31.1 Detailed Description

template<>class mlpack::kernel::KernelTraits< CosineDistance >

Kernel traits for the cosine distance.

Definition at line 56 of file cosine_distance.hpp.

21.31.2 Member Data Documentation

21.31.2.1 const bool mlpack::kernel::KernelTraits< CosineDistance >::IsNormalized = true [static]

The cosine kernel is normalized: K(x, x) = 1 for all x.

Definition at line 60 of file cosine_distance.hpp.

The documentation for this class was generated from the following file:

- src/mlpack/core/kernels/**cosine_distance.hpp**

21.32 mlpack::kernel::KernelTraits< EpanechnikovKernel > Class Template Reference

Kernel traits for the Epanechnikov kernel.

Static Public Attributes

- static const bool **IsNormalized** = true
  The Epanechnikov kernel is normalized: K(x, x) = 1 for all x.

21.32.1 Detailed Description

template<>class mlpack::kernel::KernelTraits< EpanechnikovKernel >

Kernel traits for the Epanechnikov kernel.

Definition at line 95 of file epanechnikov_kernel.hpp.

21.32.2 Member Data Documentation

21.32.2.1 const bool mlpack::kernel::KernelTraits< EpanechnikovKernel >::IsNormalized = true [static]

The Epanechnikov kernel is normalized: K(x, x) = 1 for all x.

Definition at line 99 of file epanechnikov_kernel.hpp.

The documentation for this class was generated from the following file:

- src/mlpack/core/kernels/**epanechnikov_kernel.hpp**

21.33 mlpack::kernel::KernelTraits< GaussianKernel > Class Template Reference

Kernel traits for the Gaussian kernel.

Static Public Attributes

- static const bool **IsNormalized** = true
  The Gaussian kernel is normalized: K(x, x) = 1 for all x.

21.33.1 Detailed Description

template<>class mlpack::kernel::KernelTraits< GaussianKernel >

Kernel traits for the Gaussian kernel.

Definition at line 144 of file gaussian_kernel.hpp.

21.33.2 Member Data Documentation

21.33.2.1 const bool mlpack::kernel::KernelTraits< GaussianKernel >::IsNormalized = true [static]

The Gaussian kernel is normalized: K(x, x) = 1 for all x.

Definition at line 148 of file gaussian_kernel.hpp.

The documentation for this class was generated from the following file:

- src/mlpack/core/kernels/**gaussian_kernel.hpp**

21.34 mlpack::kernel::KernelTraits< LaplacianKernel > Class Template Reference

Kernel traits of the Laplacian kernel.

Static Public Attributes

- static const bool **IsNormalized** = true

  The Laplacian kernel is normalized: K(x, x) = 1 for all x.

21.34.1 Detailed Description

template<>class mlpack::kernel::KernelTraits< LaplacianKernel >

Kernel traits of the Laplacian kernel.

Definition at line 102 of file laplacian_kernel.hpp.

21.34.2 Member Data Documentation

21.34.2.1 const bool mlpack::kernel::KernelTraits< LaplacianKernel >::IsNormalized = true [static]

The Laplacian kernel is normalized: K(x, x) = 1 for all x.

Definition at line 106 of file laplacian_kernel.hpp.

The documentation for this class was generated from the following file:

- src/mlpack/core/kernels/**laplacian_kernel.hpp**

21.35 mlpack::kernel::KernelTraits< SphericalKernel > Class Template Reference

Kernel traits for the spherical kernel.

Static Public Attributes

- static const bool **IsNormalized** = true

  The spherical kernel is normalized: K(x, x) = 1 for all x.

21.35.1 Detailed Description

template<>class mlpack::kernel::KernelTraits< SphericalKernel >

Kernel traits for the spherical kernel.

Definition at line 105 of file spherical_kernel.hpp.

21.35.2 Member Data Documentation

21.35.2.1 const bool mlpack::kernel::KernelTraits< SphericalKernel >::IsNormalized = true [static]

The spherical kernel is normalized: K(x, x) = 1 for all x.

Definition at line 109 of file spherical_kernel.hpp.

The documentation for this class was generated from the following file:

- src/mlpack/core/kernels/**spherical_kernel.hpp**

21.36 mlpack::kernel::KernelTraits< TriangularKernel > Class Template Reference

Kernel traits for the triangular kernel.

Static Public Attributes

- static const bool **IsNormalized** = true

  The triangular kernel is normalized: K(x, x) = 1 for all x.

21.36.1 Detailed Description

template<>class mlpack::kernel::KernelTraits< TriangularKernel >

Kernel traits for the triangular kernel.

Definition at line 86 of file triangular_kernel.hpp.

21.36.2 Member Data Documentation

21.36.2.1 const bool mlpack::kernel::KernelTraits< TriangularKernel >::IsNormalized = true [static]

The triangular kernel is normalized: K(x, x) = 1 for all x.

Definition at line 90 of file triangular_kernel.hpp.

The documentation for this class was generated from the following file:

- src/mlpack/core/kernels/**triangular_kernel.hpp**

21.37 mlpack::kernel::LaplacianKernel Class Reference

The standard Laplacian kernel.

Public Member Functions

- **LaplacianKernel** ()

  Default constructor; sets bandwidth to 1.0.
- **LaplacianKernel** (double **bandwidth**)

  Construct the Laplacian kernel with a custom bandwidth.
- double **Bandwidth** () const

  Get the bandwidth.
- double & **Bandwidth** ()

  Modify the bandwidth.
- template<typename VecType >
  double **Evaluate** (const VecType &a, const VecType &b) const

  Evaluation of the Laplacian kernel.
- double **Evaluate** (const double t) const

  Evaluation of the Laplacian kernel given the distance between two points.

Private Attributes

- double **bandwidth**

  Kernel bandwidth.

21.37.1 Detailed Description

The standard Laplacian kernel.

Given two vectors x, y, and a bandwidth μ (set in the constructor),

$$K(x,y) = \exp(-\frac{||x-y||}{\mu}).$$

The implementation is all in the header file because it is so simple.

Definition at line 40 of file laplacian_kernel.hpp.

21.37.2 Constructor & Destructor Documentation

21.37.2.1 mlpack::kernel::LaplacianKernel::LaplacianKernel ( ) [inline]

Default constructor; sets bandwidth to 1.0.

Definition at line 46 of file laplacian_kernel.hpp.

21.37.2.2 mlpack::kernel::LaplacianKernel::LaplacianKernel (double *bandwidth*) [inline]

Construct the Laplacian kernel with a custom bandwidth.

Parameters

bandwidth	The bandwidth of the kernel (μ).

Definition at line 54 of file laplacian_kernel.hpp.

21.37.3 Member Function Documentation

21.37.3.1 double mlpack::kernel::LaplacianKernel::Bandwidth ( ) const [inline]

Get the bandwidth.

Definition at line 91 of file laplacian_kernel.hpp.

References bandwidth.

21.37.3.2 double& mlpack::kernel::LaplacianKernel::Bandwidth ( ) [inline]

Modify the bandwidth.

Definition at line 93 of file laplacian_kernel.hpp.

References bandwidth.

21.37.3.3 template<typename VecType > double mlpack::kernel::LaplacianKernel::Evaluate (const VecType & *a,* const VecType & *b*) const [inline]

Evaluation of the Laplacian kernel.

This could be generalized to use any distance metric, not the Euclidean distance, but for now, the Euclidean distance is used.

Template Parameters

VecType	Type of vector (likely arma::vec or arma::spvec).

Parameters

a	First vector.
b	Second vector.

Returns

K(a, b) using the bandwidth (μ) specified in the constructor.

Definition at line 70 of file laplacian_kernel.hpp.

References bandwidth, and mlpack::metric::LMetric< Power, TakeRoot >::Evaluate().

21.37.3.4 double mlpack::kernel::LaplacianKernel::Evaluate (const double *t*) const [inline]

Evaluation of the Laplacian kernel given the distance between two points.

Parameters

t	The distance between the two points the kernel should be evaluated on.

Returns

K(t) using the bandwidth (μ) specified in the constructor.

Definition at line 84 of file laplacian_kernel.hpp.

References bandwidth.

21.37.4 Member Data Documentation

21.37.4.1 double mlpack::kernel::LaplacianKernel::bandwidth [private]

Kernel bandwidth.

Definition at line 97 of file laplacian_kernel.hpp.

Referenced by Bandwidth(), and Evaluate().

The documentation for this class was generated from the following file:

- src/mlpack/core/kernels/**laplacian_kernel.hpp**

21.38 mlpack::kernel::LinearKernel Class Reference

The simple linear kernel (dot product).

Public Member Functions

- **LinearKernel** ()

  This constructor does nothing; the linear kernel has no parameters to store.

Static Public Member Functions

- template<typename VecType >
  static double **Evaluate** (const VecType &a, const VecType &b)

  Simple evaluation of the dot product.

21.38.1 Detailed Description

The simple linear kernel (dot product).

For any two vectors x and y,

$$K(x,y) = x^T y$$

This kernel has no parameters and therefore the evaluation can be static.

Definition at line 42 of file linear_kernel.hpp.

21.38.2 Constructor & Destructor Documentation

21.38.2.1 mlpack::kernel::LinearKernel::LinearKernel ( ) [inline]

This constructor does nothing; the linear kernel has no parameters to store.

Definition at line 49 of file linear_kernel.hpp.

21.38.3 Member Function Documentation

21.38.3.1 template<typename VecType > static double mlpack::kernel::LinearKernel::Evaluate (const VecType & *a,* const VecType & *b*) [inline],[static]

Simple evaluation of the dot product.

This evaluation uses Armadillo's dot() function.

Template Parameters

VecType	Type of vector (should be arma::vec or arma::spvec).

Parameters

a	First vector.
b	Second vector.

Returns

K(a, b).

Definition at line 61 of file linear_kernel.hpp.

The documentation for this class was generated from the following file:

- src/mlpack/core/kernels/**linear_kernel.hpp**

21.39 mlpack::kernel::PolynomialKernel Class Reference

The simple polynomial kernel.

Public Member Functions

- **PolynomialKernel** (const double **degree**=2.0, const double **offset**=0.0)

  Construct the Polynomial Kernel with the given offset and degree.
- const double & **Degree** () const

  Get the degree of the polynomial.
- double & **Degree** ()

  Modify the degree of the polynomial.
- template<typename VecType >
  double **Evaluate** (const VecType &a, const VecType &b) const

  Simple evaluation of the dot product.
- const double & **Offset** () const

  Get the offset of the dot product of the arguments.
- double & **Offset** ()

  Modify the offset of the dot product of the arguments.

Private Attributes

- double **degree**

  The degree of the polynomial.
- double **offset**

  The offset of the dot product of the arguments.

21.39.1 Detailed Description

The simple polynomial kernel.

For any two vectors x, y, $degree$ and $offset$,

$$K(x,y) = (x^T * y + offset)^{degree}.$$

Definition at line 38 of file polynomial_kernel.hpp.

21.39.2 Constructor & Destructor Documentation

21.39.2.1 mlpack::kernel::PolynomialKernel::PolynomialKernel (const double *degree* = 2.0, const double *offset* = 0.0) [inline]

Construct the Polynomial Kernel with the given offset and degree.

If the arguments are omitted, the default degree is 2 and the default offset is 0.

Parameters

offset	Offset of the dot product of the arguments.
degree	Degree of the polynomial.

Definition at line 48 of file polynomial_kernel.hpp.

21.39.3 Member Function Documentation

21.39.3.1 const double& mlpack::kernel::PolynomialKernel::Degree ( ) const [inline]

Get the degree of the polynomial.

Definition at line 69 of file polynomial_kernel.hpp.

References degree.

21.39.3.2 double& mlpack::kernel::PolynomialKernel::Degree ( ) [inline]

Modify the degree of the polynomial.

Definition at line 71 of file polynomial_kernel.hpp.

References degree.

21.39.3.3 template<typename VecType > double mlpack::kernel::PolynomialKernel::Evaluate (const VecType & *a*, const VecType & *b*) const [inline]

Simple evaluation of the dot product.

This evaluation uses Armadillo's dot() function.

Template Parameters

VecType	Type of vector (should be arma::vec or arma::spvec).

Parameters

a	First vector.
b	Second vector.

Returns

K(a, b).

Definition at line 63 of file polynomial_kernel.hpp.

References degree, and offset.

21.39.3.4 const double& mlpack::kernel::PolynomialKernel::Offset ( ) const [inline]

Get the offset of the dot product of the arguments.

Definition at line 74 of file polynomial_kernel.hpp.

References offset.

21.39.3.5 double& mlpack::kernel::PolynomialKernel::Offset ( ) [inline]

Modify the offset of the dot product of the arguments.

Definition at line 76 of file polynomial_kernel.hpp.

References offset.

21.39.4 Member Data Documentation

21.39.4.1 double mlpack::kernel::PolynomialKernel::degree [private]

The degree of the polynomial.

Definition at line 80 of file polynomial_kernel.hpp.

Referenced by Degree(), and Evaluate().

21.39.4.2 double mlpack::kernel::PolynomialKernel::offset [private]

The offset of the dot product of the arguments.

Definition at line 82 of file polynomial_kernel.hpp.

Referenced by Evaluate(), and Offset().

The documentation for this class was generated from the following file:

- src/mlpack/core/kernels/**polynomial_kernel.hpp**

21.40 mlpack::kernel::PSpectrumStringKernel Class Reference

The p-spectrum string kernel.

Public Member Functions

- **PSpectrumStringKernel** (const std::vector< std::vector< std::string > > &**datasets**, const size_t **p**)
  *Initialize the **PSpectrumStringKernel*** (p. *252*) *with the given string datasets.*
- const std::vector< std::vector < std::map< std::string, int > > > & **Counts** () const
  Access the lists of substrings.
- std::vector< std::vector < std::map< std::string, int > > > & **Counts** ()
  Modify the lists of substrings.
- template<typename VecType >
  double **Evaluate** (const VecType &a, const VecType &b) const
  Evaluate the kernel for the string indices given.
- size_t **P** () const
  Access the value of p.
- size_t & **P** ()
  Modify the value of p.

Private Attributes

- std::vector< std::vector < std::map< std::string, int > > > **counts**
  Mappings of the datasets to counts of substrings.
- const std::vector< std::vector < std::string > > & **datasets**

The datasets.

- size_t **p**

  The value of p to use in calculation.

21.40.1 Detailed Description

The p-spectrum string kernel.

Given a length p, the p-spectrum kernel finds the contiguous subsequence match count between two strings. The kernel will take every possible substring of length p of one string and count how many times it appears in the other string.

The string kernel, when created, must be passed a reference to a series of string datasets (std::vector<std::vector<std::string> >&). This is because MLPACK only supports datasets which are Armadillo matrices – and a dataset of variable-length strings cannot be easily cast into an Armadillo matrix.

Therefore, once the **PSpectrumStringKernel** (p. 252) is created with a reference to the string datasets, a "fake" Armadillo data matrix must be created, which simply holds indices to the strings they represent. This "fake" matrix has two rows and n columns (where n is the number of strings in the dataset). The first row holds the index of the dataset (remember, the kernel can have multiple datasets), and the second row holds the index of the string. A fake matrix containing only strings from dataset 0 might look like this:

[[0 0 0 0 0 0 0 0 0] [0 1 2 3 4 5 6 7 8]]

This fake matrix is then given to the machine learning method, which will eventually call PSpectrumStringKernel::Evaluate(a, b), where a and b are two columns of the fake matrix. The string kernel will then map these fake columns back to the strings they represent, and then correctly evaluate the kernel.

Unfortunately, not every machine learning method will work with this kernel. Only machine learning methods which do not ever operate on the explicit representation of points can use this kernel. So, for instance, one cannot build a kd-tree on strings, because the BinarySpaceTree<> class will split the data according to the fake data matrix – resulting in a meaningless tree. This kernel was originally written for the FastMKS method; so, at the very least, it will work with that.

Definition at line 74 of file pspectrum_string_kernel.hpp.

21.40.2 Constructor & Destructor Documentation

21.40.2.1 mlpack::kernel::PSpectrumStringKernel::PSpectrumStringKernel (const std::vector< std::vector< std::string > > & *datasets*, const size_t *p*)

Initialize the **PSpectrumStringKernel** (p. 252) with the given string datasets.

For more information on this, see the general class documentation.

Parameters

datasets	Sets of string data.
p	The length of substrings to search.

21.40.3 Member Function Documentation

21.40.3.1 const std::vector<std::vector<std::map<std::string, int> > >& mlpack::kernel::PSpectrumStringKernel::Counts ( ) const [inline]

Access the lists of substrings.

Definition at line 102 of file pspectrum_string_kernel.hpp.

References counts.

21.40.3.2 std::vector<std::vector<std::map<std::string, int> > >& mlpack::kernel::PSpectrumStringKernel::Counts ( ) [inline]

Modify the lists of substrings.

Definition at line 105 of file pspectrum_string_kernel.hpp.

References counts.

21.40.3.3 template<typename VecType > double mlpack::kernel::PSpectrumStringKernel::Evaluate (const VecType & *a,* const VecType & *b*) const

Evaluate the kernel for the string indices given.

As mentioned in the class documentation, a and b should be 2-element vectors, where the first element contains the index of the dataset and the second element contains the index of the string. Therefore, if [2 3] is passed for a, the string used will be datasets[2][3] (datasets is of type std::vector<std::vector<std::string> >&).

Parameters

a	Index of string and dataset for first string.
b	Index of string and dataset for second string.

21.40.3.4 size_t mlpack::kernel::PSpectrumStringKernel::P ( ) const [inline]

Access the value of p.

Definition at line 109 of file pspectrum_string_kernel.hpp.

References p.

21.40.3.5 size_t& mlpack::kernel::PSpectrumStringKernel::P ( ) [inline]

Modify the value of p.

Definition at line 111 of file pspectrum_string_kernel.hpp.

References p.

21.40.4 Member Data Documentation

21.40.4.1 std::vector<std::vector<std::map<std::string, int> > > mlpack::kernel::PSpectrumStringKernel::counts [private]

Mappings of the datasets to counts of substrings.

Such a huge structure is not wonderful...

Definition at line 119 of file pspectrum_string_kernel.hpp.

Referenced by Counts().

21.40.4.2 const std::vector<std::vector<std::string> >& mlpack::kernel::PSpectrumStringKernel::datasets [private]

The datasets.

Definition at line 115 of file pspectrum_string_kernel.hpp.

21.40.4.3 size_t mlpack::kernel::PSpectrumStringKernel::p [private]

The value of p to use in calculation.

Definition at line 122 of file pspectrum_string_kernel.hpp.

Referenced by P().

The documentation for this class was generated from the following file:

- src/mlpack/core/kernels/**pspectrum_string_kernel.hpp**

21.41 mlpack::kernel::SphericalKernel Class Reference

Public Member Functions

- **SphericalKernel** ()
- **SphericalKernel** (double b)
- template<typename VecType >
  double **ConvolutionIntegral** (const VecType &a, const VecType &b)

  Obtains the convolution integral [integral K(||x-a||)K(||b-x||)dx] for the two vectors.
- template<typename VecType >
  double **Evaluate** (const VecType &a, const VecType &b)
- double **Evaluate** (double t)
- double **Normalizer** (size_t dimension)

Private Attributes

- double **bandwidth**
- double **bandwidthSquared**

21.41.1 Detailed Description

Definition at line 32 of file spherical_kernel.hpp.

21.41.2 Constructor & Destructor Documentation

21.41.2.1 mlpack::kernel::SphericalKernel::SphericalKernel ( ) [inline]

Definition at line 35 of file spherical_kernel.hpp.

21.41.2.2 mlpack::kernel::SphericalKernel::SphericalKernel (double *b*) [inline]

Definition at line 38 of file spherical_kernel.hpp.

21.41.3 Member Function Documentation

21.41.3.1 template<typename VecType > double mlpack::kernel::SphericalKernel::ConvolutionIntegral (const VecType & *a,* const VecType & *b*) [inline]

Obtains the convolution integral [integral K(||x-a||)K(||b-x||)dx] for the two vectors.

In this case, because our simple example kernel has no internal parameters, we can declare the function static. For a more complex example which cannot be declared static, see the **GaussianKernel** (p. 235), which stores an internal parameter.

Template Parameters

VecType	Type of vector (arma::vec, arma::spvec should be expected).

Parameters

a	First vector.
b	Second vector.

Returns

the convolution integral value.

Definition at line 62 of file spherical_kernel.hpp.

References bandwidth, mlpack::metric::LMetric< Power, TakeRoot >::Evaluate(), mlpack::Log::Fatal, and Normalizer().

21.41.3.2 template<typename VecType > double mlpack::kernel::SphericalKernel::Evaluate (const VecType & *a,* const VecType & *b*) [inline]

Definition at line 43 of file spherical_kernel.hpp.

References bandwidthSquared, and mlpack::metric::LMetric< Power, TakeRoot >::Evaluate().

21.41.3.3 double mlpack::kernel::SphericalKernel::Evaluate (double *t*) [inline]

Definition at line 93 of file spherical_kernel.hpp.

References bandwidth.

21.41.3.4 double mlpack::kernel::SphericalKernel::Normalizer (size_t *dimension*) [inline]

Definition at line 88 of file spherical_kernel.hpp.

References bandwidth, and M_PI.

Referenced by ConvolutionIntegral().

21.41.4 Member Data Documentation

21.41.4.1 double mlpack::kernel::SphericalKernel::bandwidth [private]

Definition at line 99 of file spherical_kernel.hpp.

Referenced by ConvolutionIntegral(), Evaluate(), and Normalizer().

21.41.4.2 double mlpack::kernel::SphericalKernel::bandwidthSquared [private]

Definition at line 100 of file spherical_kernel.hpp.

Referenced by Evaluate().

The documentation for this class was generated from the following file:

- src/mlpack/core/kernels/**spherical_kernel.hpp**

21.42 mlpack::kernel::TriangularKernel Class Reference

The trivially simple triangular kernel, defined by.

Public Member Functions

- **TriangularKernel** (const double **bandwidth**=1.0)

  Initialize the triangular kernel with the given bandwidth (default 1.0).
- double **Bandwidth** () const

  Get the bandwidth of the kernel.
- double & **Bandwidth** ()

  Modify the bandwidth of the kernel.
- template<typename Vec1Type , typename Vec2Type >
  double **Evaluate** (const Vec1Type &a, const Vec2Type &b) const

  Evaluate the triangular kernel for the two given vectors.
- double **Evaluate** (const double distance) const

  Evaluate the triangular kernel given that the distance between the two points is known.

Private Attributes

- double **bandwidth**

  The bandwidth of the kernel.

21.42.1 Detailed Description

The trivially simple triangular kernel, defined by.

$$K(x,y) = \max\{0, 1 - \frac{||x-y||_2}{b}\}$$

where b is the bandwidth of the kernel.

Definition at line 40 of file triangular_kernel.hpp.

21.42.2 Constructor & Destructor Documentation

21.42.2.1 mlpack::kernel::TriangularKernel::TriangularKernel (const double *bandwidth* = 1.0) [inline]

Initialize the triangular kernel with the given bandwidth (default 1.0).

Parameters

bandwidth	Bandwidth of the triangular kernel.

Definition at line 48 of file triangular_kernel.hpp.

21.42.3 Member Function Documentation

21.42.3.1 double mlpack::kernel::TriangularKernel::Bandwidth ( ) const [inline]

Get the bandwidth of the kernel.

Definition at line 75 of file triangular_kernel.hpp.

References bandwidth.

21.42.3.2 double& mlpack::kernel::TriangularKernel::Bandwidth ( ) [inline]

Modify the bandwidth of the kernel.

Definition at line 77 of file triangular_kernel.hpp.

References bandwidth.

21.42.3.3 template<typename Vec1Type , typename Vec2Type > double mlpack::kernel::TriangularKernel::Evaluate (const Vec1Type & *a,* const Vec2Type & *b*) const [inline]

Evaluate the triangular kernel for the two given vectors.

Parameters

a	First vector.
b	Second vector.

Definition at line 57 of file triangular_kernel.hpp.

References bandwidth, and mlpack::metric::LMetric< Power, TakeRoot >::Evaluate().

21.42.3.4 double mlpack::kernel::TriangularKernel::Evaluate (const double *distance*) const [inline]

Evaluate the triangular kernel given that the distance between the two points is known.

Parameters

distance	The distance between the two points.

Definition at line 69 of file triangular_kernel.hpp.

References bandwidth.

21.42.4 Member Data Documentation

21.42.4.1 double mlpack::kernel::TriangularKernel::bandwidth [private]

The bandwidth of the kernel.

Definition at line 81 of file triangular_kernel.hpp.

Referenced by Bandwidth(), and Evaluate().

The documentation for this class was generated from the following file:

- src/mlpack/core/kernels/**triangular_kernel.hpp**

21.43 mlpack::kmeans::AllowEmptyClusters Class Reference

Policy which allows K-Means to create empty clusters without any error being reported.

Public Member Functions

- **AllowEmptyClusters** ()
  Default constructor required by EmptyClusterPolicy policy.

Static Public Member Functions

- template<typename MatType >
  static size_t **EmptyCluster** (const MatType &, const size_t, const MatType &, arma::Col< size_t > &, arma::Col< size_t > &)
  This function does nothing.

21.43.1 Detailed Description

Policy which allows K-Means to create empty clusters without any error being reported.

Definition at line 35 of file allow_empty_clusters.hpp.

21.43.2 Constructor & Destructor Documentation

21.43.2.1 mlpack::kmeans::AllowEmptyClusters::AllowEmptyClusters ( ) [inline]

Default constructor required by EmptyClusterPolicy policy.

Definition at line 39 of file allow_empty_clusters.hpp.

21.43.3 Member Function Documentation

21.43.3.1 template<typename MatType > static size_t mlpack::kmeans::AllowEmptyClusters::EmptyCluster (const MatType & , const size_t , const MatType & , arma::Col< size_t > & , arma::Col< size_t > &) [inline], [static]

This function does nothing.

It is called by K-Means when K-Means detects an empty cluster.

Template Parameters

MatType	Type of data (arma::mat or arma::spmat).

Parameters

data	Dataset on which clustering is being performed.
emptyCluster	Index of cluster which is empty.
centroids	Centroids of each cluster (one per column).
clusterCounts	Number of points in each cluster.
assignments	Cluster assignments of each point.

Returns

Number of points changed (0).

Definition at line 55 of file allow_empty_clusters.hpp.

The documentation for this class was generated from the following file:

- src/mlpack/methods/kmeans/**allow_empty_clusters.hpp**

21.44 mlpack::kmeans::KMeans< MetricType, InitialPartitionPolicy, EmptyClusterPolicy > Class Template Reference

This class implements K-Means clustering.

Public Member Functions

- **KMeans** (const size_t **maxIterations**=1000, const double **overclusteringFactor**=1.0, const MetricType **metric**=MetricType(), const InitialPartitionPolicy **partitioner**=InitialPartitionPolicy(), const EmptyClusterPolicy **emptyClusterAction**=EmptyClusterPolicy())

  Create a K-Means object and (optionally) set the parameters which K-Means will be run with.

- template<typename MatType >
  void **Cluster** (const MatType &data, const size_t clusters, arma::Col< size_t > &assignments, const bool initialGuess=false) const

  Perform k-means clustering on the data, returning a list of cluster assignments.

- template<typename MatType >
  void **Cluster** (const MatType &data, const size_t clusters, arma::Col< size_t > &assignments, MatType &centroids, const bool initialAssignmentGuess=false, const bool initialCentroidGuess=false) const

  Perform k-means clustering on the data, returning a list of cluster assignments and also the centroids of each cluster.

- const EmptyClusterPolicy & **EmptyClusterAction** () const

  Get the empty cluster policy.

- EmptyClusterPolicy & **EmptyClusterAction** ()

  Modify the empty cluster policy.

- template<typename MatType >
  void **FastCluster** (MatType &data, const size_t clusters, arma::Col< size_t > &assignments) const

  An implementation of k-means using the Pelleg-Moore algorithm; this is known to not work – do not use it! (Fixing it is TODO, of course; see #251.)

- size_t **MaxIterations** () const

Get the maximum number of iterations.

- size_t & **MaxIterations** ()

  Set the maximum number of iterations.
- const MetricType & **Metric** () const

  Get the distance metric.
- MetricType & **Metric** ()

  Modify the distance metric.
- double **OverclusteringFactor** () const

  Return the overclustering factor.
- double & **OverclusteringFactor** ()

  Set the overclustering factor. Must be greater than 1.
- const InitialPartitionPolicy & **Partitioner** () const

  Get the initial partitioning policy.
- InitialPartitionPolicy & **Partitioner** ()

  Modify the initial partitioning policy.

Private Attributes

- EmptyClusterPolicy **emptyClusterAction**

  Instantiated empty cluster policy.
- size_t **maxIterations**

  Maximum number of iterations before giving up.
- MetricType **metric**

  Instantiated distance metric.
- double **overclusteringFactor**

  Factor controlling how many clusters are actually found.
- InitialPartitionPolicy **partitioner**

  Instantiated initial partitioning policy.

21.44.1 Detailed Description

template<typename MetricType = metric::SquaredEuclideanDistance, typename InitialPartitionPolicy = RandomPartition, typename EmptyClusterPolicy = MaxVarianceNewCluster>class mlpack::kmeans::KMeans< MetricType, InitialPartitionPolicy, EmptyClusterPolicy >

This class implements K-Means clustering.

This implementation supports overclustering, which means that more clusters than are requested will be found; then, those clusters will be merged together to produce the desired number of clusters.

Two template parameters can (optionally) be supplied: the policy for how to find the initial partition of the data, and the actions to be taken when an empty cluster is encountered, as well as the distance metric to be used.

A simple example of how to run K-Means clustering is shown below.

```
extern arma::mat data; // Dataset we want to run K-Means on.
arma::Col<size_t> assignments; // Cluster assignments.

KMeans<> k; // Default options.
k.Cluster(data, 3, assignments); // 3 clusters.

// Cluster using the Manhattan distance, 100 iterations maximum, and an
// overclustering factor of 4.0.
KMeans<metric::ManhattanDistance> k(100, 4.0);
k.Cluster(data, 6, assignments); // 6 clusters.
```

Template Parameters

MetricType	The distance metric to use for this **KMeans** (p. 260); see **metric::LMetric** (p. 288) for an example.
InitialPartitionPolicy	Initial partitioning policy; must implement a default constructor and 'void Cluster(const arma::mat&, const size_t, arma::Col<size_t>&)'.
EmptyClusterPolicy	Policy for what to do on an empty cluster; must implement a default constructor and 'void EmptyCluster(const arma::mat&, arma::Col<size_t&)'.

See Also

RandomPartition (p. 267), **RefinedStart** (p. 269), **AllowEmptyClusters** (p. 259), **MaxVarianceNewCluster** (p. 266)

Definition at line 75 of file kmeans.hpp.

21.44.2 Constructor & Destructor Documentation

21.44.2.1 **template<typename MetricType = metric::SquaredEuclideanDistance, typename InitialPartitionPolicy = RandomPartition, typename EmptyClusterPolicy = MaxVarianceNewCluster> mlpack::kmeans::KMeans< MetricType, InitialPartitionPolicy, EmptyClusterPolicy >::KMeans (const size_t *maxIterations* = `1000`, const double *overclusteringFactor* = `1.0`, const MetricType *metric* = `MetricType()`, const InitialPartitionPolicy *partitioner* = `InitialPartitionPolicy()`, const EmptyClusterPolicy *emptyClusterAction* = `EmptyClusterPolicy()`)**

Create a K-Means object and (optionally) set the parameters which K-Means will be run with.

This implementation allows a few strategies to improve the performance of K-Means, including "overclustering" and disallowing empty clusters.

The overclustering factor controls how many clusters are actually found; for instance, with an overclustering factor of 4, if K-Means is run to find 3 clusters, it will actually find 12, then merge the nearest clusters until only 3 are left.

Parameters

maxIterations	Maximum number of iterations allowed before giving up (0 is valid, but the algorithm may never terminate).
overclustering-Factor	Factor controlling how many extra clusters are found and then merged to get the desired number of clusters.
metric	Optional MetricType object; for when the metric has state it needs to store.
partitioner	Optional InitialPartitionPolicy object; for when a specially initialized partitioning policy is required.
emptyCluster-Action	Optional EmptyClusterPolicy object; for when a specially initialized empty cluster policy is required.

21.44.3 Member Function Documentation

21.44.3.1 **template<typename MetricType = metric::SquaredEuclideanDistance, typename InitialPartitionPolicy = RandomPartition, typename EmptyClusterPolicy = MaxVarianceNewCluster> template<typename MatType > void mlpack::kmeans::KMeans< MetricType, InitialPartitionPolicy, EmptyClusterPolicy >::Cluster (const MatType & *data*, const size_t *clusters*, arma::Col< size_t > & *assignments*, const bool *initialGuess* = `false`) const**

Perform k-means clustering on the data, returning a list of cluster assignments.

Optionally, the vector of assignments can be set to an initial guess of the cluster assignments; to do this, set initialGuess to true.

Template Parameters

MatType	Type of matrix (arma::mat or arma::sp_mat).

Parameters

data	Dataset to cluster.
clusters	Number of clusters to compute.
assignments	Vector to store cluster assignments in.
initialGuess	If true, then it is assumed that assignments has a list of initial cluster assignments.

21.44.3.2 template<typename MetricType = metric::SquaredEuclideanDistance, typename InitialPartitionPolicy = RandomPartition, typename EmptyClusterPolicy = MaxVarianceNewCluster> template<typename MatType > void mlpack::kmeans::KMeans< MetricType, InitialPartitionPolicy, EmptyClusterPolicy >::Cluster (const MatType & *data*, const size_t *clusters*, arma::Col< size_t > & *assignments*, MatType & *centroids*, const bool *initialAssignmentGuess* = `false`, const bool *initialCentroidGuess* = `false`) const

Perform k-means clustering on the data, returning a list of cluster assignments and also the centroids of each cluster.

Optionally, the vector of assignments can be set to an initial guess of the cluster assignments; to do this, set initialAssignmentGuess to true. Another way to set initial cluster guesses is to fill the centroids matrix with the centroid guesses, and then set initialCentroidGuess to true. initialAssignmentGuess supersedes initialCentroidGuess, so if both are set to true, the assignments vector is used.

Note that if the overclustering factor is greater than 1, the centroids matrix will be resized in the method. Regardless of the overclustering factor, the centroid guess matrix (if initialCentroidGuess is set to true) should have the same number of rows as the data matrix, and number of columns equal to 'clusters'.

Template Parameters

MatType	Type of matrix (arma::mat or arma::sp_mat).

Parameters

data	Dataset to cluster.
clusters	Number of clusters to compute.
assignments	Vector to store cluster assignments in.
centroids	Matrix in which centroids are stored.
initialAssignment-Guess	If true, then it is assumed that assignments has a list of initial cluster assignments.
initialCentroid-Guess	If true, then it is assumed that centroids contains the initial centroids of each cluster.

21.44.3.3 template<typename MetricType = metric::SquaredEuclideanDistance, typename InitialPartitionPolicy = RandomPartition, typename EmptyClusterPolicy = MaxVarianceNewCluster> const EmptyClusterPolicy& mlpack::kmeans::KMeans< MetricType, InitialPartitionPolicy, EmptyClusterPolicy >::EmptyClusterAction ( ) const `[inline]`

Get the empty cluster policy.

Definition at line 191 of file kmeans.hpp.

References mlpack::kmeans::KMeans< MetricType, InitialPartitionPolicy, EmptyClusterPolicy >::emptyClusterAction.

21.44.3.4 template<typename MetricType = metric::SquaredEuclideanDistance, typename InitialPartitionPolicy = RandomPartition, typename EmptyClusterPolicy = MaxVarianceNewCluster> EmptyClusterPolicy& mlpack::kmeans::KMeans< MetricType, InitialPartitionPolicy, EmptyClusterPolicy >::EmptyClusterAction ( ) [inline]

Modify the empty cluster policy.

Definition at line 194 of file kmeans.hpp.

References mlpack::kmeans::KMeans< MetricType, InitialPartitionPolicy, EmptyClusterPolicy >::emptyClusterAction.

21.44.3.5 template<typename MetricType = metric::SquaredEuclideanDistance, typename InitialPartitionPolicy = RandomPartition, typename EmptyClusterPolicy = MaxVarianceNewCluster> template<typename MatType > void mlpack::kmeans::KMeans< MetricType, InitialPartitionPolicy, EmptyClusterPolicy >::FastCluster (MatType & *data,* const size_t *clusters,* arma::Col< size_t > & *assignments*) const

An implementation of k-means using the Pelleg-Moore algorithm; this is known to not work – do not use it! (Fixing it is TODO, of course; see #251.)

21.44.3.6 template<typename MetricType = metric::SquaredEuclideanDistance, typename InitialPartitionPolicy = RandomPartition, typename EmptyClusterPolicy = MaxVarianceNewCluster> size_t mlpack::kmeans::KMeans< MetricType, InitialPartitionPolicy, EmptyClusterPolicy >::MaxIterations ( ) const [inline]

Get the maximum number of iterations.

Definition at line 176 of file kmeans.hpp.

References mlpack::kmeans::KMeans< MetricType, InitialPartitionPolicy, EmptyClusterPolicy >::maxIterations.

21.44.3.7 template<typename MetricType = metric::SquaredEuclideanDistance, typename InitialPartitionPolicy = RandomPartition, typename EmptyClusterPolicy = MaxVarianceNewCluster> size_t& mlpack::kmeans::KMeans< MetricType, InitialPartitionPolicy, EmptyClusterPolicy >::MaxIterations ( ) [inline]

Set the maximum number of iterations.

Definition at line 178 of file kmeans.hpp.

References mlpack::kmeans::KMeans< MetricType, InitialPartitionPolicy, EmptyClusterPolicy >::maxIterations.

21.44.3.8 template<typename MetricType = metric::SquaredEuclideanDistance, typename InitialPartitionPolicy = RandomPartition, typename EmptyClusterPolicy = MaxVarianceNewCluster> const MetricType& mlpack::kmeans::KMeans< MetricType, InitialPartitionPolicy, EmptyClusterPolicy >::Metric ( ) const [inline]

Get the distance metric.

Definition at line 181 of file kmeans.hpp.

References mlpack::kmeans::KMeans< MetricType, InitialPartitionPolicy, EmptyClusterPolicy >::metric.

21.44.3.9 template<typename MetricType = metric::SquaredEuclideanDistance, typename InitialPartitionPolicy = RandomPartition, typename EmptyClusterPolicy = MaxVarianceNewCluster> MetricType& mlpack::kmeans::KMeans< MetricType, InitialPartitionPolicy, EmptyClusterPolicy >::Metric ( ) [inline]

Modify the distance metric.

Definition at line 183 of file kmeans.hpp.

References mlpack::kmeans::KMeans< MetricType, InitialPartitionPolicy, EmptyClusterPolicy >::metric.

21.44.3.10 template<typename MetricType = metric::SquaredEuclideanDistance, typename InitialPartitionPolicy = RandomPartition, typename EmptyClusterPolicy = MaxVarianceNewCluster> double mlpack::kmeans::KMeans< MetricType, InitialPartitionPolicy, EmptyClusterPolicy >::OverclusteringFactor ( ) const [inline]

Return the overclustering factor.

Definition at line 171 of file kmeans.hpp.

References mlpack::kmeans::KMeans< MetricType, InitialPartitionPolicy, EmptyClusterPolicy >::overclusteringFactor.

21.44.3.11 template<typename MetricType = metric::SquaredEuclideanDistance, typename InitialPartitionPolicy = RandomPartition, typename EmptyClusterPolicy = MaxVarianceNewCluster> double& mlpack::kmeans::KMeans< MetricType, InitialPartitionPolicy, EmptyClusterPolicy >::OverclusteringFactor ( ) [inline]

Set the overclustering factor. Must be greater than 1.

Definition at line 173 of file kmeans.hpp.

References mlpack::kmeans::KMeans< MetricType, InitialPartitionPolicy, EmptyClusterPolicy >::overclusteringFactor.

21.44.3.12 template<typename MetricType = metric::SquaredEuclideanDistance, typename InitialPartitionPolicy = RandomPartition, typename EmptyClusterPolicy = MaxVarianceNewCluster> const InitialPartitionPolicy& mlpack::kmeans::KMeans< MetricType, InitialPartitionPolicy, EmptyClusterPolicy >::Partitioner ( ) const [inline]

Get the initial partitioning policy.

Definition at line 186 of file kmeans.hpp.

References mlpack::kmeans::KMeans< MetricType, InitialPartitionPolicy, EmptyClusterPolicy >::partitioner.

21.44.3.13 template<typename MetricType = metric::SquaredEuclideanDistance, typename InitialPartitionPolicy = RandomPartition, typename EmptyClusterPolicy = MaxVarianceNewCluster> InitialPartitionPolicy& mlpack::kmeans::KMeans< MetricType, InitialPartitionPolicy, EmptyClusterPolicy >::Partitioner ( ) [inline]

Modify the initial partitioning policy.

Definition at line 188 of file kmeans.hpp.

References mlpack::kmeans::KMeans< MetricType, InitialPartitionPolicy, EmptyClusterPolicy >::partitioner.

21.44.4 Member Data Documentation

21.44.4.1 template<typename MetricType = metric::SquaredEuclideanDistance, typename InitialPartitionPolicy = RandomPartition, typename EmptyClusterPolicy = MaxVarianceNewCluster> EmptyClusterPolicy mlpack::kmeans::KMeans< MetricType, InitialPartitionPolicy, EmptyClusterPolicy >::emptyClusterAction [private]

Instantiated empty cluster policy.

Definition at line 206 of file kmeans.hpp.

Referenced by mlpack::kmeans::KMeans< MetricType, InitialPartitionPolicy, EmptyClusterPolicy >::EmptyClusterAction().

21.44.4.2 template<typename MetricType = metric::SquaredEuclideanDistance, typename InitialPartitionPolicy = RandomPartition, typename EmptyClusterPolicy = MaxVarianceNewCluster> size_t mlpack::kmeans::KMeans< MetricType, InitialPartitionPolicy, EmptyClusterPolicy >::maxIterations [private]

Maximum number of iterations before giving up.

Definition at line 200 of file kmeans.hpp.

Referenced by mlpack::kmeans::KMeans< MetricType, InitialPartitionPolicy, EmptyClusterPolicy >::MaxIterations().

21.44.4.3 template<typename MetricType = metric::SquaredEuclideanDistance, typename InitialPartitionPolicy = RandomPartition, typename EmptyClusterPolicy = MaxVarianceNewCluster> MetricType mlpack::kmeans::KMeans< MetricType, InitialPartitionPolicy, EmptyClusterPolicy >::metric [private]

Instantiated distance metric.

Definition at line 202 of file kmeans.hpp.

Referenced by mlpack::kmeans::KMeans< MetricType, InitialPartitionPolicy, EmptyClusterPolicy >::Metric().

21.44.4.4 template<typename MetricType = metric::SquaredEuclideanDistance, typename InitialPartitionPolicy = RandomPartition, typename EmptyClusterPolicy = MaxVarianceNewCluster> double mlpack::kmeans::KMeans< MetricType, InitialPartitionPolicy, EmptyClusterPolicy >::overclusteringFactor [private]

Factor controlling how many clusters are actually found.

Definition at line 198 of file kmeans.hpp.

Referenced by mlpack::kmeans::KMeans< MetricType, InitialPartitionPolicy, EmptyClusterPolicy >::OverclusteringFactor().

21.44.4.5 template<typename MetricType = metric::SquaredEuclideanDistance, typename InitialPartitionPolicy = RandomPartition, typename EmptyClusterPolicy = MaxVarianceNewCluster> InitialPartitionPolicy mlpack::kmeans::KMeans< MetricType, InitialPartitionPolicy, EmptyClusterPolicy >::partitioner [private]

Instantiated initial partitioning policy.

Definition at line 204 of file kmeans.hpp.

Referenced by mlpack::kmeans::KMeans< MetricType, InitialPartitionPolicy, EmptyClusterPolicy >::Partitioner().

The documentation for this class was generated from the following file:

- src/mlpack/methods/kmeans/**kmeans.hpp**

21.45 mlpack::kmeans::MaxVarianceNewCluster Class Reference

When an empty cluster is detected, this class takes the point furthest from the centroid of the cluster with maximum variance as a new cluster.

Public Member Functions

- **MaxVarianceNewCluster** ()

  Default constructor required by EmptyClusterPolicy.

Static Public Member Functions

- template<typename MatType >
  static size_t **EmptyCluster** (const MatType &data, const size_t emptyCluster, const MatType &centroids, arma::-Col< size_t > &clusterCounts, arma::Col< size_t > &assignments)

  Take the point furthest from the centroid of the cluster with maximum variance to be a new cluster.

21.45.1 Detailed Description

When an empty cluster is detected, this class takes the point furthest from the centroid of the cluster with maximum variance as a new cluster.

Definition at line 36 of file max_variance_new_cluster.hpp.

21.45.2 Constructor & Destructor Documentation

21.45.2.1 mlpack::kmeans::MaxVarianceNewCluster::MaxVarianceNewCluster ( ) [inline]

Default constructor required by EmptyClusterPolicy.

Definition at line 40 of file max_variance_new_cluster.hpp.

21.45.3 Member Function Documentation

21.45.3.1 template<typename MatType > static size_t mlpack::kmeans::MaxVarianceNewCluster::EmptyCluster (const MatType & *data,* const size_t *emptyCluster,* const MatType & *centroids,* arma::Col< size_t > & *clusterCounts,* arma::Col< size_t > & *assignments*) [static]

Take the point furthest from the centroid of the cluster with maximum variance to be a new cluster.

Template Parameters

MatType	Type of data (arma::mat or arma::sp_mat).

Parameters

data	Dataset on which clustering is being performed.
emptyCluster	Index of cluster which is empty.
centroids	Centroids of each cluster (one per column).
clusterCounts	Number of points in each cluster.
assignments	Cluster assignments of each point.

Returns

Number of points changed.

The documentation for this class was generated from the following file:

- src/mlpack/methods/kmeans/**max_variance_new_cluster.hpp**

21.46 mlpack::kmeans::RandomPartition Class Reference

A very simple partitioner which partitions the data randomly into the number of desired clusters.

Public Member Functions

- **RandomPartition** ()

  Empty constructor, required by the InitialPartitionPolicy policy.

Static Public Member Functions

- template<typename MatType >
  static void **Cluster** (const MatType &data, const size_t clusters, arma::Col< size_t > &assignments)

  Partition the given dataset into the given number of clusters.

21.46.1 Detailed Description

A very simple partitioner which partitions the data randomly into the number of desired clusters.

It has no parameters, and so an instance of the class is not even necessary.

Definition at line 36 of file random_partition.hpp.

21.46.2 Constructor & Destructor Documentation

21.46.2.1 mlpack::kmeans::RandomPartition::RandomPartition ( ) [inline]

Empty constructor, required by the InitialPartitionPolicy policy.

Definition at line 40 of file random_partition.hpp.

21.46.3 Member Function Documentation

21.46.3.1 template<typename MatType > static void mlpack::kmeans::RandomPartition::Cluster (const MatType & *data,* const size_t *clusters,* arma::Col< size_t > & *assignments*) [inline], [static]

Partition the given dataset into the given number of clusters.

Assignments are random, and the number of points in each cluster should be equal (or approximately equal).

Template Parameters

MatType	Type of data (arma::mat or arma::sp_mat).

Parameters

data	Dataset to partition.
clusters	Number of clusters to split dataset into.
assignments	Vector to store cluster assignments into. Values will be between 0 and (clusters - 1).

Definition at line 54 of file random_partition.hpp.

The documentation for this class was generated from the following file:

- src/mlpack/methods/kmeans/**random_partition.hpp**

21.47 mlpack::kmeans::RefinedStart Class Reference

A refined approach for choosing initial points for k-means clustering.

Public Member Functions

- **RefinedStart** (const size_t **samplings**=100, const double **percentage**=0.02)

  *Create the **RefinedStart** (p. 269) object, optionally specifying parameters for the number of samplings to perform and the percentage of the dataset to use in each sampling.*
- template<typename MatType >
  void **Cluster** (const MatType &data, const size_t clusters, arma::Col< size_t > &assignments) const

  Partition the given dataset into the given number of clusters according to the random sampling scheme outlined in Bradley and Fayyad's paper.
- double **Percentage** () const

  Get the percentage of the data used by each subsampling.
- double & **Percentage** ()

  Modify the percentage of the data used by each subsampling.
- size_t **Samplings** () const

  Get the number of samplings that will be performed.
- size_t & **Samplings** ()

  Modify the number of samplings that will be performed.

Private Attributes

- double **percentage**

  The percentage of the data to use for each subsampling.
- size_t **samplings**

  The number of samplings to perform.

21.47.1 Detailed Description

A refined approach for choosing initial points for k-means clustering.

This approach runs k-means several times on random subsets of the data, and then clusters those solutions to select refined initial cluster assignments. It is an implementation of the following paper:

{bradley1998refining, title={Refining initial points for k-means clustering}, author={Bradley, Paul S and Fayyad, Usama M}, booktitle={Proceedings of the Fifteenth International Conference on Machine Learning (ICML 1998)}, volume={66}, year={1998} }

Definition at line 47 of file refined_start.hpp.

21.47.2 Constructor & Destructor Documentation

21.47.2.1 mlpack::kmeans::RefinedStart::RefinedStart (const size_t *samplings* = 100, const double *percentage* = 0.02) [inline]

Create the **RefinedStart** (p. 269) object, optionally specifying parameters for the number of samplings to perform and the percentage of the dataset to use in each sampling.

Definition at line 55 of file refined_start.hpp.

21.47.3 Member Function Documentation

21.47.3.1 template<typename MatType > void mlpack::kmeans::RefinedStart::Cluster (const MatType & *data,* const size_t *clusters,* arma::Col< size_t > & *assignments*) const

Partition the given dataset into the given number of clusters according to the random sampling scheme outlined in Bradley and Fayyad's paper.

Template Parameters

MatType	Type of data (arma::mat or arma::sp_mat).

Parameters

data	Dataset to partition.
clusters	Number of clusters to split dataset into.
assignments	Vector to store cluster assignments into. Values will be between 0 and (clusters - 1).

21.47.3.2 double mlpack::kmeans::RefinedStart::Percentage ( ) const [inline]

Get the percentage of the data used by each subsampling.

Definition at line 80 of file refined_start.hpp.

References percentage.

21.47.3.3 double& mlpack::kmeans::RefinedStart::Percentage ( ) [inline]

Modify the percentage of the data used by each subsampling.

Definition at line 82 of file refined_start.hpp.

References percentage.

21.47.3.4 size_t mlpack::kmeans::RefinedStart::Samplings ( ) const [inline]

Get the number of samplings that will be performed.

Definition at line 75 of file refined_start.hpp.

References samplings.

21.47.3.5 size_t& mlpack::kmeans::RefinedStart::Samplings ( ) [inline]

Modify the number of samplings that will be performed.

Definition at line 77 of file refined_start.hpp.

References samplings.

21.47.4 Member Data Documentation

21.47.4.1 double mlpack::kmeans::RefinedStart::percentage [private]

The percentage of the data to use for each subsampling.

Definition at line 88 of file refined_start.hpp.

Referenced by Percentage().

21.47.4.2 size_t mlpack::kmeans::RefinedStart::samplings [private]

The number of samplings to perform.

Definition at line 86 of file refined_start.hpp.

Referenced by Samplings().

The documentation for this class was generated from the following file:

- src/mlpack/methods/kmeans/**refined_start.hpp**

21.48 mlpack::kpca::KernelPCA< KernelType > Class Template Reference

This class performs kernel principal components analysis (Kernel PCA), for a given kernel.

Public Member Functions

- **KernelPCA** (const KernelType **kernel**=KernelType(), const bool **centerTransformedData**=false)
  *Construct the **KernelPCA** (p. 271) object, optionally passing a kernel.*
- void **Apply** (const arma::mat &data, arma::mat &transformedData, arma::vec &eigval, arma::mat &eigvec)
  Apply Kernel Principal Components Analysis to the provided data set.
- void **Apply** (const arma::mat &data, arma::mat &transformedData, arma::vec &eigval)
  Apply Kernel Principal Component Analysis to the provided data set.
- void **Apply** (arma::mat &data, const size_t newDimension)
  Apply dimensionality reduction using Kernel Principal Component Analysis to the provided data set.
- bool **CenterTransformedData** () const
  Return whether or not the transformed data is centered.
- bool & **CenterTransformedData** ()
  Return whether or not the transformed data is centered.
- const KernelType & **Kernel** () const
  Get the kernel.
- KernelType & **Kernel** ()
  Modify the kernel.

Private Member Functions

- void **GetKernelMatrix** (const arma::mat &data, arma::mat &kernelMatrix)
  Construct the kernel matrix.

Private Attributes

- bool **centerTransformedData**
  *If true, the data will be scaled (by standard deviation) when **Apply()** (p. 272) is run.*
- KernelType **kernel**
  The instantiated kernel.

21.48.1 Detailed Description

template<typename KernelType>class mlpack::kpca::KernelPCA< KernelType >

This class performs kernel principal components analysis (Kernel PCA), for a given kernel.

This is a standard machine learning technique and is well-documented on the Internet and in standard texts. It is often used as a dimensionality reduction technique, and can also be useful in mapping linearly inseparable classes of points to different spaces where they are linearly separable.

The performance of the method is highly dependent on the kernel choice. There are numerous available kernels in the **mlpack::kernel** (p. 93) namespace (see files in mlpack/core/kernels/) and it is easy to write your own; see other implementations for examples.

Definition at line 46 of file kernel_pca.hpp.

21.48.2 Constructor & Destructor Documentation

21.48.2.1 template<typename KernelType > mlpack::kpca::KernelPCA< KernelType >::KernelPCA (const KernelType *kernel* = `KernelType()`, const bool *centerTransformedData* = `false`)

Construct the **KernelPCA** (p. 271) object, optionally passing a kernel.

Optionally, the transformed data can be centered about the origin; to do this, pass 'true' for centerTransformedData. This will take slightly longer (but not much).

Parameters

kernel	Kernel to be used for computation.

21.48.3 Member Function Documentation

21.48.3.1 template<typename KernelType > void mlpack::kpca::KernelPCA< KernelType >::Apply (const arma::mat & *data*, arma::mat & *transformedData*, arma::vec & *eigval*, arma::mat & *eigvec*)

Apply Kernel Principal Components Analysis to the provided data set.

Parameters

data	Data matrix.
transformedData	Matrix to output results into.
eigval	KPCA eigenvalues will be written to this vector.
eigvec	KPCA eigenvectors will be written to this matrix.

21.48.3.2 template<typename KernelType > void mlpack::kpca::KernelPCA< KernelType >::Apply (const arma::mat & *data*, arma::mat & *transformedData*, arma::vec & *eigval*)

Apply Kernel Principal Component Analysis to the provided data set.

Parameters

data	Data matrix.
transformedData	Matrix to output results into.
eigval	KPCA eigenvalues will be written to this vector.

21.48.3.3 template<typename KernelType > void mlpack::kpca::KernelPCA< KernelType >::Apply (arma::mat & *data,* const size_t *newDimension*)

Apply dimensionality reduction using Kernel Principal Component Analysis to the provided data set.

The data matrix will be modified in-place. Note that the dimension can be larger than the existing dimension because KPCA works on the kernel matrix, not the covariance matrix. This means the new dimension can be as large as the number of points (columns) in the dataset. Note that if you specify newDimension to be larger than the current dimension of the data (the number of rows), then it's not really "dimensionality reduction"...

Parameters

data	Data matrix.
newDimension	New dimension for the dataset.

21.48.3.4 template<typename KernelType > bool mlpack::kpca::KernelPCA< KernelType >::CenterTransformedData ( ) const [inline]

Return whether or not the transformed data is centered.

Definition at line 105 of file kernel_pca.hpp.

References mlpack::kpca::KernelPCA< KernelType >::centerTransformedData.

21.48.3.5 template<typename KernelType > bool& mlpack::kpca::KernelPCA< KernelType >::CenterTransformedData ( ) [inline]

Return whether or not the transformed data is centered.

Definition at line 107 of file kernel_pca.hpp.

References mlpack::kpca::KernelPCA< KernelType >::centerTransformedData.

21.48.3.6 template<typename KernelType > void mlpack::kpca::KernelPCA< KernelType >::GetKernelMatrix (const arma::mat & *data,* arma::mat & *kernelMatrix*) [private]

Construct the kernel matrix.

Parameters

data	Input data points.
kernelMatrix	Matrix to store the constructed kernel matrix in.

21.48.3.7 template<typename KernelType > const KernelType& mlpack::kpca::KernelPCA< KernelType >::Kernel ( ) const [inline]

Get the kernel.

Definition at line 100 of file kernel_pca.hpp.

References mlpack::kpca::KernelPCA< KernelType >::kernel.

21.48.3.8 template<typename KernelType > KernelType& mlpack::kpca::KernelPCA< KernelType >::Kernel ( ) [inline]

Modify the kernel.

Definition at line 102 of file kernel_pca.hpp.

References mlpack::kpca::KernelPCA< KernelType >::kernel.

21.48.4 Member Data Documentation

21.48.4.1 template<typename KernelType > bool mlpack::kpca::KernelPCA< KernelType >::centerTransformedData [private]

If true, the data will be scaled (by standard deviation) when **Apply()** (p. 272) is run.

Definition at line 114 of file kernel_pca.hpp.

Referenced by mlpack::kpca::KernelPCA< KernelType >::CenterTransformedData().

21.48.4.2 template<typename KernelType > KernelType mlpack::kpca::KernelPCA< KernelType >::kernel [private]

The instantiated kernel.

Definition at line 111 of file kernel_pca.hpp.

Referenced by mlpack::kpca::KernelPCA< KernelType >::Kernel().

The documentation for this class was generated from the following file:

- src/mlpack/methods/kernel_pca/**kernel_pca.hpp**

21.49 mlpack::lcc::LocalCoordinateCoding< DictionaryInitializer > Class Template Reference

An implementation of Local Coordinate Coding (LCC) that codes data which approximately lives on a manifold using a variation of l1-norm regularized sparse coding; in LCC, the penalty on the absolute value of each point's coefficient for each atom is weighted by the squared distance of that point to that atom.

Public Member Functions

- **LocalCoordinateCoding** (const arma::mat &**data**, const size_t **atoms**, const double **lambda**)

  *Set the parameters to **LocalCoordinateCoding** (p. 274).*
- const arma::mat & **Codes** () const

  Accessor the codes.
- arma::mat & **Codes** ()

  Modify the codes.
- const arma::mat & **Data** () const

  Access the data.

- const arma::mat & **Dictionary** () const

  Accessor for dictionary.
- arma::mat & **Dictionary** ()

  Mutator for dictionary.
- void **Encode** (const size_t maxIterations=0, const double objTolerance=0.01)

  Run local coordinate coding.
- double **Objective** (arma::uvec adjacencies) const

  Compute objective function given the list of adjacencies.
- void **OptimizeCode** ()

  Code each point via distance-weighted LARS.
- void **OptimizeDictionary** (arma::uvec adjacencies)

  Learn dictionary by solving linear system.

Private Attributes

- size_t **atoms**

  Number of atoms in dictionary.
- arma::mat **codes**

  Codes (columns are points).
- const arma::mat & **data**

  Data matrix (columns are points).
- arma::mat **dictionary**

  Dictionary (columns are atoms).
- double **lambda**

  l1 regularization term.

21.49.1 Detailed Description

template<typename DictionaryInitializer = sparse_coding::DataDependentRandomInitializer>class mlpack::lcc::LocalCoordinateCoding< DictionaryInitializer >

An implementation of Local Coordinate Coding (LCC) that codes data which approximately lives on a manifold using a variation of l1-norm regularized sparse coding; in LCC, the penalty on the absolute value of each point's coefficient for each atom is weighted by the squared distance of that point to that atom.

Let d be the number of dimensions in the original space, m the number of training points, and k the number of atoms in the dictionary (the dimension of the learned feature space). The training data X is a d-by-m matrix where each column is a point and each row is a dimension. The dictionary D is a d-by-k matrix, and the sparse codes matrix Z is a k-by-m matrix. This program seeks to minimize the objective: min_{D,Z} ||X - D Z||_{Fro}^2

- lambda sum_{i=1}^m sum_{j=1}^k dist(X_i,D_j)^2 Z_i^j where lambda > 0.

This problem is solved by an algorithm that alternates between a dictionary learning step and a sparse coding step. The dictionary learning step updates the dictionary D by solving a linear system (note that the objective is a positive definite quadratic program). The sparse coding step involves solving a large number of weighted l1-norm regularized linear regression problems problems; this can be done efficiently using LARS, an algorithm that can solve the LASSO (paper below).

The papers are listed below.

```
@incollection{NIPS2009_0719,
  title = {Nonlinear Learning using Local Coordinate Coding},
  author = {Kai Yu and Tong Zhang and Yihong Gong},
  booktitle = {Advances in Neural Information Processing Systems 22},
  editor = {Y. Bengio and D. Schuurmans and J. Lafferty and C. K. I. Williams
      and A. Culotta},
  pages = {2223--2231},
  year = {2009}
}


@article{efron2004least,
  title={Least angle regression},
  author={Efron, B. and Hastie, T. and Johnstone, I. and Tibshirani, R.},
  journal={The Annals of statistics},
  volume={32},
  number={2},
  pages={407--499},
  year={2004},
  publisher={Institute of Mathematical Statistics}
}
```

Definition at line 91 of file lcc.hpp.

21.49.2 Constructor & Destructor Documentation

21.49.2.1 template<typename DictionaryInitializer = sparse_coding::DataDependentRandomInitializer> mlpack::lcc::LocalCoordinateCoding< DictionaryInitializer >::LocalCoordinateCoding (const arma::mat & *data*, const size_t *atoms*, const double *lambda*)

Set the parameters to **LocalCoordinateCoding** (p. 274).

Parameters

data	Data matrix.
atoms	Number of atoms in dictionary.
lambda	Regularization parameter for weighted l1-norm penalty.

21.49.3 Member Function Documentation

21.49.3.1 template<typename DictionaryInitializer = sparse_coding::DataDependentRandomInitializer> const arma::mat& mlpack::lcc::LocalCoordinateCoding< DictionaryInitializer >::Codes ( ) const [inline]

Accessor the codes.

Definition at line 144 of file lcc.hpp.

References mlpack::lcc::LocalCoordinateCoding< DictionaryInitializer >::codes.

21.49.3.2 template<typename DictionaryInitializer = sparse_coding::DataDependentRandomInitializer> arma::mat& mlpack::lcc::LocalCoordinateCoding< DictionaryInitializer >::Codes ( ) [inline]

Modify the codes.

Definition at line 146 of file lcc.hpp.

References mlpack::lcc::LocalCoordinateCoding< DictionaryInitializer >::codes.

21.49.3.3 template<typename DictionaryInitializer = sparse_coding::DataDependentRandomInitializer> const arma::mat& mlpack::lcc::LocalCoordinateCoding< DictionaryInitializer >::Data ( ) const [inline]

Access the data.

Definition at line 136 of file lcc.hpp.

References mlpack::lcc::LocalCoordinateCoding< DictionaryInitializer >::data.

21.49.3.4 template<typename DictionaryInitializer = sparse_coding::DataDependentRandomInitializer> const arma::mat& mlpack::lcc::LocalCoordinateCoding< DictionaryInitializer >::Dictionary ( ) const [inline]

Accessor for dictionary.

Definition at line 139 of file lcc.hpp.

References mlpack::lcc::LocalCoordinateCoding< DictionaryInitializer >::dictionary.

21.49.3.5 template<typename DictionaryInitializer = sparse_coding::DataDependentRandomInitializer> arma::mat& mlpack::lcc::LocalCoordinateCoding< DictionaryInitializer >::Dictionary ( ) [inline]

Mutator for dictionary.

Definition at line 141 of file lcc.hpp.

References mlpack::lcc::LocalCoordinateCoding< DictionaryInitializer >::dictionary.

21.49.3.6 template<typename DictionaryInitializer = sparse_coding::DataDependentRandomInitializer> void mlpack::lcc::LocalCoordinateCoding< DictionaryInitializer >::Encode (const size_t *maxIterations* = 0, const double *objTolerance* = 0.01)

Run local coordinate coding.

Parameters

nIterations	Maximum number of iterations to run algorithm.
objTolerance	Tolerance of objective function. When the objective function changes by a value lower than this tolerance, the optimization terminates.

21.49.3.7 template<typename DictionaryInitializer = sparse_coding::DataDependentRandomInitializer> double mlpack::lcc::LocalCoordinateCoding< DictionaryInitializer >::Objective (arma::uvec *adjacencies*) const

Compute objective function given the list of adjacencies.

21.49.3.8 template<typename DictionaryInitializer = sparse_coding::DataDependentRandomInitializer> void mlpack::lcc::LocalCoordinateCoding< DictionaryInitializer >::OptimizeCode ( )

Code each point via distance-weighted LARS.

21.49.3.9 template<typename DictionaryInitializer = sparse_coding::DataDependentRandomInitializer> void mlpack::lcc::LocalCoordinateCoding< DictionaryInitializer >::OptimizeDictionary (arma::uvec *adjacencies*)

Learn dictionary by solving linear system.

Parameters

adjacencies	Indices of entries (unrolled column by column) of the coding matrix Z that are non-zero (the adjacency matrix for the bipartite graph of points and atoms)

21.49.4 Member Data Documentation

21.49.4.1 template<typename DictionaryInitializer = sparse_coding::DataDependentRandomInitializer> size_t mlpack::lcc::LocalCoordinateCoding< DictionaryInitializer >::atoms [private]

Number of atoms in dictionary.

Definition at line 150 of file lcc.hpp.

21.49.4.2 template<typename DictionaryInitializer = sparse_coding::DataDependentRandomInitializer> arma::mat mlpack::lcc::LocalCoordinateCoding< DictionaryInitializer >::codes [private]

Codes (columns are points).

Definition at line 159 of file lcc.hpp.

Referenced by mlpack::lcc::LocalCoordinateCoding< DictionaryInitializer >::Codes().

21.49.4.3 template<typename DictionaryInitializer = sparse_coding::DataDependentRandomInitializer> const arma::mat& mlpack::lcc::LocalCoordinateCoding< DictionaryInitializer >::data [private]

Data matrix (columns are points).

Definition at line 153 of file lcc.hpp.

Referenced by mlpack::lcc::LocalCoordinateCoding< DictionaryInitializer >::Data().

21.49.4.4 template<typename DictionaryInitializer = sparse_coding::DataDependentRandomInitializer> arma::mat mlpack::lcc::LocalCoordinateCoding< DictionaryInitializer >::dictionary [private]

Dictionary (columns are atoms).

Definition at line 156 of file lcc.hpp.

Referenced by mlpack::lcc::LocalCoordinateCoding< DictionaryInitializer >::Dictionary().

21.49.4.5 template<typename DictionaryInitializer = sparse_coding::DataDependentRandomInitializer> double mlpack::lcc::LocalCoordinateCoding< DictionaryInitializer >::lambda [private]

l1 regularization term.

Definition at line 162 of file lcc.hpp.

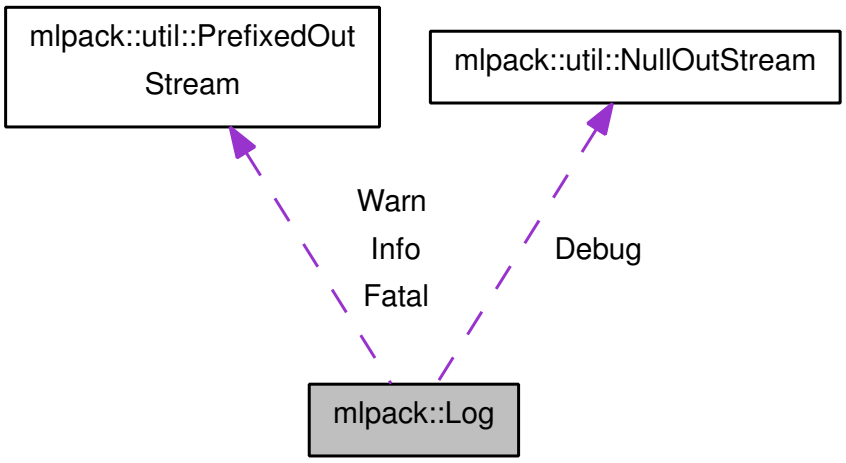
The documentation for this class was generated from the following file:

- src/mlpack/methods/local_coordinate_coding/**lcc.hpp**

21.50 mlpack::Log Class Reference

Provides a convenient way to give formatted output.

Collaboration diagram for mlpack::Log:



Static Public Member Functions

- static void **Assert** (bool condition, const std::string &message="Assert Failed.")

  Checks if the specified condition is true.

Static Public Attributes

- static std::ostream & **cout**

  Reference to cout, if necessary.
- static **util::NullOutStream Debug**

  Dumps debug output into the bit nether regions.
- static **util::PrefixedOutStream Fatal**

  Prints fatal messages prefixed with [FATAL], then terminates the program.
- static **util::PrefixedOutStream Info**

  Prints informational messages if –verbose is specified, prefixed with [INFO].
- static **util::PrefixedOutStream Warn**

  Prints warning messages prefixed with [WARN].

21.50.1 Detailed Description

Provides a convenient way to give formatted output.

The **Log** (p. 279) class has four members which can be used in the same way ostreams can be used:

- **Log::Debug** (p. 280)
- **Log::Info** (p. 281)

- **Log::Warn** (p. 281)
- **Log::Fatal** (p. 281)

Each of these will prefix a tag to the output (for easy filtering), and the fatal output will terminate the program when a newline is encountered. An example is given below.

```
Log::Info << "Checking a condition." << std::endl;
if (!someCondition())
  Log::Warn << "someCondition() is not satisfied!" << std::endl;
Log::Info << "Checking an important condition." << std::endl;
if (!someImportantCondition())
{
  Log::Fatal << "someImportantCondition() is not satisfied! Terminating.";
  Log::Fatal << std::endl;
}
```

Any messages sent to **Log::Debug** (p. 280) will not be shown when compiling in non-debug mode. Messages to **Log::Info** (p. 281) will only be shown when the –verbose flag is given to the program (or rather, the **CLI** (p. 132) class).

See Also

PrefixedOutStream, NullOutStream, **CLI** (p. 132)

Definition at line 65 of file log.hpp.

21.50.2 Member Function Documentation

21.50.2.1 static void mlpack::Log::Assert (bool *condition*, const std::string & *message* = "Assert Failed.") [static]

Checks if the specified condition is true.

If not, halts program execution and prints a custom error message. Does nothing in non-debug mode.

Referenced by mlpack::emst::EdgePair::EdgePair().

21.50.3 Member Data Documentation

21.50.3.1 std::ostream& mlpack::Log::cout [static]

Reference to cout, if necessary.

Definition at line 98 of file log.hpp.

21.50.3.2 util::NullOutStream mlpack::Log::Debug [static]

Dumps debug output into the bit nether regions.

Definition at line 84 of file log.hpp.

Referenced by mlpack::gmm::PositiveDefiniteConstraint::ApplyConstraint(), mlpack::gmm::GMM< FittingType >::GMM(), and mlpack::distribution::DiscreteDistribution::Probability().

21.50.3.3 util::PrefixedOutStream mlpack::Log::Fatal [static]

Prints fatal messages prefixed with [FATAL], then terminates the program.

Definition at line 95 of file log.hpp.

Referenced by mlpack::kernel::SphericalKernel::ConvolutionIntegral(), and mlpack::gmm::EigenvalueRatioConstraint::-EigenvalueRatioConstraint().

21.50.3.4 util::PrefixedOutStream mlpack::Log::Info [static]

Prints informational messages if –verbose is specified, prefixed with [INFO].

Definition at line 89 of file log.hpp.

21.50.3.5 util::PrefixedOutStream mlpack::Log::Warn [static]

Prints warning messages prefixed with [WARN].

Definition at line 92 of file log.hpp.

Referenced by mlpack::gmm::EigenvalueRatioConstraint::EigenvalueRatioConstraint(), mlpack::nmf::RandomAcol-Initialization< p >::Initialize(), mlpack::cf::CF::NumRecs(), and mlpack::cf::CF::NumUsersForSimilarity().

The documentation for this class was generated from the following file:

- src/mlpack/core/util/**log.hpp**

21.51 mlpack::math::Range Class Reference

Simple real-valued range.

Public Member Functions

- **Range** ()

  The upper bound.
- **Range** (const double point)
- **Range** (const double **lo**, const double **hi**)

  Initializes to specified range.
- bool **Contains** (const double d) const

  Determines if a point is contained within the range.
- bool **Contains** (const **Range** &r) const

  Determines if another range overlaps with this one.
- double **Hi** () const

  Get the upper bound.
- double & **Hi** ()

  Modify the upper bound.
- double **Lo** () const

  Get the lower bound.
- double & **Lo** ()

  Modify the lower bound.

- double **Mid** () const

  Gets the midpoint of this range.
- bool **operator!=** (const **Range** &rhs) const

  Compare with another range for strict equality.
- **Range operator&** (const **Range** &rhs) const

  Shrinks this range to be the overlap with another range; this makes an empty set if there is no overlap.
- **Range** & **operator&=** (const **Range** &rhs)

  Shrinks this range to be the overlap with another range; this makes an empty set if there is no overlap.
- **Range operator**∗ (const double d) const

  Scale the bounds by the given double.
- **Range** & **operator∗=** (const double d)

  Scale the bounds by the given double.
- bool **operator**< (const **Range** &rhs) const

  Compare with another range.
- bool **operator==** (const **Range** &rhs) const

  Compare with another range for strict equality.
- bool **operator**> (const **Range** &rhs) const

  Compare with another range.
- **Range operator|** (const **Range** &rhs) const

  Expands this range to include another range.
- **Range** & **operator|=** (const **Range** &rhs)

  Expands this range to include another range.
- std::string **ToString** () const

  Returns a string representation of an object.
- double **Width** () const

  Gets the span of the range (hi - lo).

Private Attributes

- double **hi**

  The lower bound.
- double **lo**

Friends

- **Range operator**∗ (const double d, const **Range** &r)

  Scale the bounds by the given double.

21.51.1 Detailed Description

Simple real-valued range.

It contains an upper and lower bound.

Definition at line 31 of file range.hpp.

21.51.2 Constructor & Destructor Documentation

21.51.2.1 mlpack::math::Range::Range ( ) [inline]

The upper bound.

Initialize to an empty set (where lo $>$ hi).

21.51.2.2 mlpack::math::Range::Range (const double *point*) [inline]

21.51.2.3 mlpack::math::Range::Range (const double *lo*, const double *hi*) [inline]

Initializes to specified range.

Parameters

lo	Lower bound of the range.
hi	Upper bound of the range.

21.51.3 Member Function Documentation

21.51.3.1 bool mlpack::math::Range::Contains (const double *d*) const [inline]

Determines if a point is contained within the range.

Parameters

d	Point to check.

21.51.3.2 bool mlpack::math::Range::Contains (const Range & *r*) const [inline]

Determines if another range overlaps with this one.

Parameters

r	Other range.

Returns

true if ranges overlap at all.

21.51.3.3 double mlpack::math::Range::Hi ( ) const [inline]

Get the upper bound.

Definition at line 63 of file range.hpp.

References hi.

21.51.3.4 double& mlpack::math::Range::Hi ( ) [inline]

Modify the upper bound.

Definition at line 65 of file range.hpp.

References hi.

21.51.3.5 double mlpack::math::Range::Lo ( ) const [inline]

Get the lower bound.

Definition at line 58 of file range.hpp.

References lo.

21.51.3.6 double& mlpack::math::Range::Lo ( ) [inline]

Modify the lower bound.

Definition at line 60 of file range.hpp.

References lo.

21.51.3.7 double mlpack::math::Range::Mid ( ) const [inline]

Gets the midpoint of this range.

21.51.3.8 bool mlpack::math::Range::operator!= (const Range & *rhs*) const [inline]

Compare with another range for strict equality.

Parameters

rhs	Other range.

21.51.3.9 Range mlpack::math::Range::operator& (const Range & *rhs*) const [inline]

Shrinks this range to be the overlap with another range; this makes an empty set if there is no overlap.

Parameters

rhs	Other range.

21.51.3.10 Range& mlpack::math::Range::operator&= (const Range & *rhs*) [inline]

Shrinks this range to be the overlap with another range; this makes an empty set if there is no overlap.

Parameters

rhs	Other range.

21.51.3.11 Range mlpack::math::Range::operator* (const double *d*) const [inline]

Scale the bounds by the given double.

Parameters

d	Scaling factor.

21.51.3.12 Range& mlpack::math::Range::operator*= (const double *d*) [inline]

Scale the bounds by the given double.

Parameters

d	Scaling factor.

21.51.3.13 bool mlpack::math::Range::operator< (const Range & *rhs*) const [inline]

Compare with another range.

For **Range** (p. 281) objects x and y, x $<$ y means that x is strictly less than y and does not overlap at all.

Parameters

rhs	Other range.

21.51.3.14 bool mlpack::math::Range::operator== (const Range & *rhs*) const [inline]

Compare with another range for strict equality.

Parameters

rhs	Other range.

21.51.3.15 bool mlpack::math::Range::operator> (const Range & *rhs*) const [inline]

Compare with another range.

For **Range** (p. 281) objects x and y, x $<$ y means that x is strictly less than y and does not overlap at all.

Parameters

rhs	Other range.

21.51.3.16 Range mlpack::math::Range::operator| (const Range & *rhs*) const [inline]

Expands this range to include another range.

Parameters

rhs	**Range** (p. 281) to include.

21.51.3.17 Range& mlpack::math::Range::operator|= (const Range & *rhs*) [inline]

Expands this range to include another range.

Parameters

rhs	**Range** (p. 281) to include.

21.51.3.18 std::string mlpack::math::Range::ToString ( ) const [inline]

Returns a string representation of an object.

21.51.3.19 double mlpack::math::Range::Width ( ) const [inline]

Gets the span of the range (hi - lo).

21.51.4 Friends And Related Function Documentation

21.51.4.1 Range operator∗ (const double *d*, const Range & *r*) [friend]

Scale the bounds by the given double.

Parameters

d	Scaling factor.

21.51.5 Member Data Documentation

21.51.5.1 double mlpack::math::Range::hi [private]

The lower bound.

Definition at line 35 of file range.hpp.

Referenced by Hi().

21.51.5.2 double mlpack::math::Range::lo [private]

Definition at line 34 of file range.hpp.

Referenced by Lo().

The documentation for this class was generated from the following file:

- src/mlpack/core/math/**range.hpp**

21.52 mlpack::metric::IPMetric< KernelType > Class Template Reference

Public Member Functions

- **IPMetric** ()
  *Create the **IPMetric** (p. 286) without an instantiated kernel.*
- **IPMetric** (KernelType &**kernel**)
  *Create the **IPMetric** (p. 286) with an instantiated kernel.*

- ~**IPMetric** ()

  *Destroy the **IPMetric** (p. 286) object.*

- template<typename Vec1Type , typename Vec2Type >
  double **Evaluate** (const Vec1Type &a, const Vec2Type &b)

  Evaluate the metric.

- const KernelType & **Kernel** () const

  Get the kernel.

- KernelType & **Kernel** ()

  Modify the kernel.

Private Attributes

- KernelType & **kernel**

  The reference to the kernel that is being used.

- KernelType * **localKernel**

  The locally stored kernel, if it is necessary.

21.52.1 Detailed Description

template<typename KernelType>class mlpack::metric::IPMetric< KernelType >

Definition at line 30 of file ip_metric.hpp.

21.52.2 Constructor & Destructor Documentation

21.52.2.1 template<typename KernelType > mlpack::metric::IPMetric< KernelType >::IPMetric ( )

Create the **IPMetric** (p. 286) without an instantiated kernel.

21.52.2.2 template<typename KernelType > mlpack::metric::IPMetric< KernelType >::IPMetric (KernelType & *kernel*)

Create the **IPMetric** (p. 286) with an instantiated kernel.

21.52.2.3 template<typename KernelType > mlpack::metric::IPMetric< KernelType >::~IPMetric ( )

Destroy the **IPMetric** (p. 286) object.

21.52.3 Member Function Documentation

21.52.3.1 template<typename KernelType > template<typename Vec1Type , typename Vec2Type > double mlpack::metric::IPMetric< KernelType >::Evaluate (const Vec1Type & *a,* const Vec2Type & *b*)

Evaluate the metric.

21.52.3.2 template<typename KernelType > const KernelType& mlpack::metric::IPMetric< KernelType >::Kernel ( ) const [inline]

Get the kernel.

Definition at line 49 of file ip_metric.hpp.

References mlpack::metric::IPMetric< KernelType >::kernel.

21.52.3.3 template<typename KernelType > KernelType& mlpack::metric::IPMetric< KernelType >::Kernel ( ) [inline]

Modify the kernel.

Definition at line 51 of file ip_metric.hpp.

References mlpack::metric::IPMetric< KernelType >::kernel.

21.52.4 Member Data Documentation

21.52.4.1 template<typename KernelType > KernelType& mlpack::metric::IPMetric< KernelType >::kernel [private]

The reference to the kernel that is being used.

Definition at line 57 of file ip_metric.hpp.

Referenced by mlpack::metric::IPMetric< KernelType >::Kernel().

21.52.4.2 template<typename KernelType > KernelType* mlpack::metric::IPMetric< KernelType >::localKernel [private]

The locally stored kernel, if it is necessary.

Definition at line 55 of file ip_metric.hpp.

The documentation for this class was generated from the following file:

- src/mlpack/core/metrics/**ip_metric.hpp**

21.53 mlpack::metric::LMetric< Power, TakeRoot > Class Template Reference

The L_p metric for arbitrary integer p, with an option to take the root.

Public Member Functions

- **LMetric** ()

Static Public Member Functions

- template<typename VecType1 , typename VecType2 >
  static double **Evaluate** (const VecType1 &a, const VecType2 &b)

  Computes the distance between two points.

21.53.1 Detailed Description

template<int Power, bool TakeRoot = true>class mlpack::metric::LMetric< Power, TakeRoot >

The L_p metric for arbitrary integer p, with an option to take the root.

This class implements the standard L_p metric for two arbitrary vectors x and y of dimensionality n:

$$d(x,y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{\frac{1}{p}}.$$

The value of p is given as a template parameter.

In addition, the function $d(x,y)$ can be simplified, neglecting the p-root calculation. This is done by specifying the TakeRoot template parameter to be false. Then,

$$d(x,y) = \sum_{i=1}^{n} |x_i - y_i|^p$$

It is faster to compute that distance, so TakeRoot is by default off. However, when TakeRoot is false, the distance given is not actually a true metric – it does not satisfy the triangle inequality. Some MLPACK methods do not require the triangle inequality to operate correctly (such as the BinarySpaceTree), but setting TakeRoot = false in some cases will cause incorrect results.

A few convenience typedefs are given:

- ManhattanDistance
- EuclideanDistance
- SquaredEuclideanDistance

Template Parameters

Power	Power of metric; i.e. Power = 1 gives the L1-norm (Manhattan distance).
TakeRoot	If true, the Power'th root of the result is taken before it is returned. Setting this to false causes the metric to not satisfy the Triangle Inequality (be careful!).

Definition at line 73 of file lmetric.hpp.

21.53.2 Constructor & Destructor Documentation

21.53.2.1 template<int Power, bool TakeRoot = true> mlpack::metric::LMetric< Power, TakeRoot >::LMetric ( ) [inline]

Definition at line 80 of file lmetric.hpp.

21.53.3 Member Function Documentation

21.53.3.1 template<int Power, bool TakeRoot = true> template<typename VecType1 , typename VecType2 > static double mlpack::metric::LMetric< Power, TakeRoot >::Evaluate (const VecType1 & *a,* const VecType2 & *b*) [static]

Computes the distance between two points.

Referenced by mlpack::kernel::SphericalKernel::ConvolutionIntegral(), mlpack::kernel::GaussianKernel::ConvolutionIntegral(), mlpack::kernel::SphericalKernel::Evaluate(), mlpack::kernel::TriangularKernel::Evaluate(), mlpack::kernel::LaplacianKernel::Evaluate(), and mlpack::kernel::GaussianKernel::Evaluate().

The documentation for this class was generated from the following file:

- src/mlpack/core/metrics/**lmetric.hpp**

21.54 mlpack::metric::MahalanobisDistance< t_take_root > Class Template Reference

The Mahalanobis distance, which is essentially a stretched Euclidean distance.

Public Member Functions

- **MahalanobisDistance** ()

  Initialize the Mahalanobis distance with the empty matrix as covariance.
- **MahalanobisDistance** (const size_t dimensionality)

  Initialize the Mahalanobis distance with the identity matrix of the given dimensionality.
- **MahalanobisDistance** (const arma::mat &**covariance**)

  Initialize the Mahalanobis distance with the given covariance matrix.
- const arma::mat & **Covariance** () const

  Access the covariance matrix.
- arma::mat & **Covariance** ()

  Modify the covariance matrix.
- template<typename VecType1 , typename VecType2 >
  double **Evaluate** (const VecType1 &a, const VecType2 &b)

  Evaluate the distance between the two given points using this Mahalanobis distance.

Private Attributes

- arma::mat **covariance**

  The covariance matrix associated with this distance.

21.54.1 Detailed Description

template<bool t_take_root = false>class mlpack::metric::MahalanobisDistance< t_take_root >

The Mahalanobis distance, which is essentially a stretched Euclidean distance.

Given a square covariance matrix Q of size d x d, where d is the dimensionality of the points it will be evaluating, and given two vectors x and y also of dimensionality d,

$$d(x,y) = \sqrt{(x-y)^T Q (x-y)}$$

where Q is the covariance matrix.

Because each evaluation multiplies (x_1 - x_2) by the covariance matrix, it may be much quicker to use an **LMetric** (p. 288) and simply stretch the actual dataset itself before performing any evaluations. However, this class is provided for convenience.

Similar to the **LMetric** (p. 288) class, this offers a template parameter t_take_root which, when set to false, will instead evaluate the distance

$$d(x,y) = (x-y)^T Q(x-y)$$

which is faster to evaluate.

Template Parameters

t_take_root	If true, takes the root of the output. It is slightly faster to leave this at the default of false.

Definition at line 61 of file mahalanobis_distance.hpp.

21.54.2 Constructor & Destructor Documentation

21.54.2.1 template<bool t_take_root = false> mlpack::metric::MahalanobisDistance< t_take_root >::MahalanobisDistance ( ) [inline]

Initialize the Mahalanobis distance with the empty matrix as covariance.

Don't call **Evaluate()** (p. 292) until you set the covariance with **Covariance()** (p. 292)!

Definition at line 68 of file mahalanobis_distance.hpp.

21.54.2.2 template<bool t_take_root = false> mlpack::metric::MahalanobisDistance< t_take_root >::MahalanobisDistance (const size_t *dimensionality*) [inline]

Initialize the Mahalanobis distance with the identity matrix of the given dimensionality.

Parameters

dimensionality	Dimesnsionality of the covariance matrix.

Definition at line 76 of file mahalanobis_distance.hpp.

21.54.2.3 template<bool t_take_root = false> mlpack::metric::MahalanobisDistance< t_take_root >::MahalanobisDistance (const arma::mat & *covariance*) [inline]

Initialize the Mahalanobis distance with the given covariance matrix.

The given covariance matrix will be copied (this is not optimal).

Parameters

covariance	The covariance matrix to use for this distance.

Definition at line 85 of file mahalanobis_distance.hpp.

21.54.3 Member Function Documentation

21.54.3.1 template<bool t_take_root = false> const arma::mat& mlpack::metric::MahalanobisDistance< t_take_root >::Covariance ( ) const [inline]

Access the covariance matrix.

Returns

Constant reference to the covariance matrix.

Definition at line 104 of file mahalanobis_distance.hpp.

References mlpack::metric::MahalanobisDistance< t_take_root >::covariance.

21.54.3.2 template<bool t_take_root = false> arma::mat& mlpack::metric::MahalanobisDistance< t_take_root >::Covariance ( ) [inline]

Modify the covariance matrix.

Returns

Reference to the covariance matrix.

Definition at line 111 of file mahalanobis_distance.hpp.

References mlpack::metric::MahalanobisDistance< t_take_root >::covariance.

21.54.3.3 template<bool t_take_root = false> template<typename VecType1 , typename VecType2 > double mlpack::metric::MahalanobisDistance< t_take_root >::Evaluate (const VecType1 & *a,* const VecType2 & *b*)

Evaluate the distance between the two given points using this Mahalanobis distance.

If the covariance matrix has not been set (i.e. if you used the empty constructor and did not later modify the covariance matrix), calling this method will probably result in a crash.

Parameters

a	First vector.
b	Second vector.

21.54.4 Member Data Documentation

21.54.4.1 template<bool t_take_root = false> arma::mat mlpack::metric::MahalanobisDistance< t_take_root >::covariance [private]

The covariance matrix associated with this distance.

Definition at line 115 of file mahalanobis_distance.hpp.

Referenced by mlpack::metric::MahalanobisDistance< t_take_root >::Covariance().

The documentation for this class was generated from the following file:

- src/mlpack/core/metrics/**mahalanobis_distance.hpp**

21.55 mlpack::naive_bayes::NaiveBayesClassifier< MatType > Class Template Reference

The simple Naive Bayes classifier.

Public Member Functions

- **NaiveBayesClassifier** (const MatType &data, const arma::Col< size_t > &labels, const size_t classes)

  Initializes the classifier as per the input and then trains it by calculating the sample mean and variances.
- void **Classify** (const MatType &data, arma::Col< size_t > &results)

  Given a bunch of data points, this function evaluates the class of each of those data points, and puts it in the vector 'results'.
- const MatType & **Means** () const

  Get the sample means for each class.
- MatType & **Means** ()

  Modify the sample means for each class.
- const arma::vec & **Probabilities** () const

  Get the prior probabilities for each class.
- arma::vec & **Probabilities** ()

  Modify the prior probabilities for each class.
- const MatType & **Variances** () const

  Get the sample variances for each class.
- MatType & **Variances** ()

  Modify the sample variances for each class.

Private Attributes

- MatType **means**

  Sample mean for each class.
- arma::vec **probabilities**

  Class probabilities.
- MatType **variances**

  Sample variances for each class.

21.55.1 Detailed Description

template<typename MatType = arma::mat>class mlpack::naive_bayes::NaiveBayesClassifier< MatType >

The simple Naive Bayes classifier.

This class trains on the data by calculating the sample mean and variance of the features with respect to each of the labels, and also the class probabilities. The class labels are assumed to be positive integers (starting with 0), and are expected to be the last row of the data input to the constructor.

Mathematically, it computes P(X_i = x_i | Y = y_j) for each feature X_i for each of the labels y_j. Alongwith this, it also computes the classs probabilities P(Y = y_j).

For classifying a data point (x_1, x_2, ..., x_n), it computes the following: arg max_y(P(Y = y)*P(X_1 = x_1 | Y = y) * ... * P(X_n = x_n | Y = y))

Example use:

```
extern arma::mat training_data, testing_data;
NaiveBayesClassifier<> nbc(training_data, 5);
arma::vec results;

nbc.Classify(testing_data, results);
```

Definition at line 58 of file naive_bayes_classifier.hpp.

21.55.2 Constructor & Destructor Documentation

21.55.2.1 template<typename MatType = arma::mat> mlpack::naive_bayes::NaiveBayesClassifier< MatType >::NaiveBayesClassifier (const MatType & *data*, const arma::Col< size_t > & *labels*, const size_t *classes*)

Initializes the classifier as per the input and then trains it by calculating the sample mean and variances.

The input data is expected to have integer labels as the last row (starting with 0 and not greater than the number of classes).

Example use:

```
extern arma::mat training_data, testing_data;
extern arma::Col<size_t> labels;
NaiveBayesClassifier nbc(training_data, labels, 5);
```

Parameters

data	Training data points.
labels	Labels corresponding to training data points.
classes	Number of classes in this classifier.

21.55.3 Member Function Documentation

21.55.3.1 template<typename MatType = arma::mat> void mlpack::naive_bayes::NaiveBayesClassifier< MatType >::Classify (const MatType & *data*, arma::Col< size_t > & *results*)

Given a bunch of data points, this function evaluates the class of each of those data points, and puts it in the vector 'results'.

```
arma::mat test_data; // each column is a test point
arma::Col<size_t> results;
...
nbc.Classify(test_data, &results);
```

Parameters

data	List of data points.
results	Vector that class predictions will be placed into.

21.55.3.2 template<typename MatType = arma::mat> const MatType& mlpack::naive_bayes::NaiveBayesClassifier< MatType >::Means ( ) const [inline]

Get the sample means for each class.

Definition at line 109 of file naive_bayes_classifier.hpp.

References mlpack::naive_bayes::NaiveBayesClassifier< MatType >::means.

21.55.3.3 template<typename MatType = arma::mat> MatType& mlpack::naive_bayes::NaiveBayesClassifier< MatType >::Means ( ) [inline]

Modify the sample means for each class.

Definition at line 111 of file naive_bayes_classifier.hpp.

References mlpack::naive_bayes::NaiveBayesClassifier< MatType >::means.

21.55.3.4 template<typename MatType = arma::mat> const arma::vec& mlpack::naive_bayes::NaiveBayesClassifier< MatType >::Probabilities ( ) const [inline]

Get the prior probabilities for each class.

Definition at line 119 of file naive_bayes_classifier.hpp.

References mlpack::naive_bayes::NaiveBayesClassifier< MatType >::probabilities.

21.55.3.5 template<typename MatType = arma::mat> arma::vec& mlpack::naive_bayes::NaiveBayesClassifier< MatType >::Probabilities ( ) [inline]

Modify the prior probabilities for each class.

Definition at line 121 of file naive_bayes_classifier.hpp.

References mlpack::naive_bayes::NaiveBayesClassifier< MatType >::probabilities.

21.55.3.6 template<typename MatType = arma::mat> const MatType& mlpack::naive_bayes::NaiveBayesClassifier< MatType >::Variances ( ) const [inline]

Get the sample variances for each class.

Definition at line 114 of file naive_bayes_classifier.hpp.

References mlpack::naive_bayes::NaiveBayesClassifier< MatType >::variances.

21.55.3.7 template<typename MatType = arma::mat> MatType& mlpack::naive_bayes::NaiveBayesClassifier< MatType >::Variances ( ) [inline]

Modify the sample variances for each class.

Definition at line 116 of file naive_bayes_classifier.hpp.

References mlpack::naive_bayes::NaiveBayesClassifier< MatType >::variances.

21.55.4 Member Data Documentation

21.55.4.1 template<typename MatType = arma::mat> MatType mlpack::naive_bayes::NaiveBayesClassifier< MatType >::means [private]

Sample mean for each class.

Definition at line 62 of file naive_bayes_classifier.hpp.

Referenced by mlpack::naive_bayes::NaiveBayesClassifier< MatType >::Means().

21.55.4.2 template<typename MatType = arma::mat> arma::vec mlpack::naive_bayes::NaiveBayesClassifier< MatType >::probabilities [private]

Class probabilities.

Definition at line 68 of file naive_bayes_classifier.hpp.

Referenced by mlpack::naive_bayes::NaiveBayesClassifier< MatType >::Probabilities().

21.55.4.3 template<typename MatType = arma::mat> MatType mlpack::naive_bayes::NaiveBayesClassifier< MatType >::variances [private]

Sample variances for each class.

Definition at line 65 of file naive_bayes_classifier.hpp.

Referenced by mlpack::naive_bayes::NaiveBayesClassifier< MatType >::Variances().

The documentation for this class was generated from the following file:

- src/mlpack/methods/naive_bayes/**naive_bayes_classifier.hpp**

21.56 mlpack::nca::NCA< MetricType, OptimizerType > Class Template Reference

An implementation of Neighborhood Components Analysis, both a linear dimensionality reduction technique and a distance learning technique.

Public Member Functions

- **NCA** (const arma::mat &**dataset**, const arma::Col< size_t > &**labels**, MetricType **metric**=MetricType())
  Construct the Neighborhood Components Analysis object.
- const arma::mat & **Dataset** () const
  Get the dataset reference.
- const arma::Col< size_t > & **Labels** () const
  Get the labels reference.
- void **LearnDistance** (arma::mat &outputMatrix)
  Perform Neighborhood Components Analysis.
- const OptimizerType < **SoftmaxErrorFunction** < MetricType > > & **Optimizer** () const
  Get the optimizer.
- OptimizerType < **SoftmaxErrorFunction** < MetricType > > & **Optimizer** ()

Private Attributes

- const arma::mat & **dataset**
  Dataset reference.
- **SoftmaxErrorFunction**< MetricType > **errorFunction**
  The function to optimize.
- const arma::Col< size_t > & **labels**
  Labels reference.
- MetricType **metric**
  Metric to be used.
- OptimizerType < **SoftmaxErrorFunction** < MetricType > > **optimizer**
  The optimizer to use.

21.56.1 Detailed Description

template<typename MetricType = metric::SquaredEuclideanDistance, template< typename > class OptimizerType = optimization::SGD>class mlpack::nca::NCA< MetricType, OptimizerType >

An implementation of Neighborhood Components Analysis, both a linear dimensionality reduction technique and a distance learning technique.

The method seeks to improve k-nearest-neighbor classification on a dataset by scaling the dimensions. The method is nonparametric, and does not require a value of k. It works by using stochastic ("soft") neighbor assignments and using optimization techniques over the gradient of the accuracy of the neighbor assignments.

For more details, see the following published paper:

```
@inproceedings{Goldberger2004,
  author = {Goldberger, Jacob and Roweis, Sam and Hinton, Geoff and
      Salakhutdinov, Ruslan},
  booktitle = {Advances in Neural Information Processing Systems 17},
  pages = {513--520},
  publisher = {MIT Press},
  title = {{Neighbourhood Components Analysis}},
  year = {2004}
}
```

Definition at line 59 of file nca.hpp.

21.56.2 Constructor & Destructor Documentation

21.56.2.1 template<typename MetricType = metric::SquaredEuclideanDistance, template< typename > class OptimizerType = optimization::SGD> mlpack::nca::NCA< MetricType, OptimizerType >::NCA (const arma::mat & *dataset,* const arma::Col< size_t > & *labels,* MetricType *metric* = `MetricType()`)

Construct the Neighborhood Components Analysis object.

This simply stores the reference to the dataset and labels as well as the parameters for optimization before the actual optimization is performed.

Parameters

dataset	Input dataset.
labels	Input dataset labels.
stepSize	Step size for stochastic gradient descent.
maxIterations	Maximum iterations for stochastic gradient descent.
tolerance	Tolerance for termination of stochastic gradient descent.
shuffle	Whether or not to shuffle the dataset during SGD.
metric	Instantiated metric to use.

21.56.3 Member Function Documentation

21.56.3.1 template<typename MetricType = metric::SquaredEuclideanDistance, template< typename > class OptimizerType = optimization::SGD> const arma::mat& mlpack::nca::NCA< MetricType, OptimizerType >::Dataset ( ) const `[inline]`

Get the dataset reference.

Definition at line 91 of file nca.hpp.

References mlpack::nca::NCA< MetricType, OptimizerType >::dataset.

21.56.3.2 template<typename MetricType = metric::SquaredEuclideanDistance, template< typename > class OptimizerType = optimization::SGD> const arma::Col<size_t>& mlpack::nca::NCA< MetricType, OptimizerType >::Labels ( ) const [inline]

Get the labels reference.

Definition at line 93 of file nca.hpp.

References mlpack::nca::NCA< MetricType, OptimizerType >::labels.

21.56.3.3 template<typename MetricType = metric::SquaredEuclideanDistance, template< typename > class OptimizerType = optimization::SGD> void mlpack::nca::NCA< MetricType, OptimizerType >::LearnDistance (arma::mat & *outputMatrix*)

Perform Neighborhood Components Analysis.

The output distance learning matrix is written into the passed reference. If **LearnDistance()** (p. 298) is called with an outputMatrix which has the correct size (dataset.n_rows x dataset.n_rows), that matrix will be used as the starting point for optimization.

Parameters

output_matrix	Covariance matrix of Mahalanobis distance.

21.56.3.4 template<typename MetricType = metric::SquaredEuclideanDistance, template< typename > class OptimizerType = optimization::SGD> const OptimizerType<SoftmaxErrorFunction<MetricType> >& mlpack::nca::NCA< MetricType, OptimizerType >::Optimizer ( ) const [inline]

Get the optimizer.

Definition at line 96 of file nca.hpp.

References mlpack::nca::NCA< MetricType, OptimizerType >::optimizer.

21.56.3.5 template<typename MetricType = metric::SquaredEuclideanDistance, template< typename > class OptimizerType = optimization::SGD> OptimizerType<SoftmaxErrorFunction<MetricType> >& mlpack::nca::NCA< MetricType, OptimizerType >::Optimizer ( ) [inline]

Definition at line 98 of file nca.hpp.

References mlpack::nca::NCA< MetricType, OptimizerType >::optimizer.

21.56.4 Member Data Documentation

21.56.4.1 template<typename MetricType = metric::SquaredEuclideanDistance, template< typename > class OptimizerType = optimization::SGD> const arma::mat& mlpack::nca::NCA< MetricType, OptimizerType >::dataset [private]

Dataset reference.

Definition at line 103 of file nca.hpp.

Referenced by mlpack::nca::NCA< MetricType, OptimizerType >::Dataset().

21.56.4.2 template<typename MetricType = metric::SquaredEuclideanDistance, template< typename > class OptimizerType = optimization::SGD> SoftmaxErrorFunction<MetricType> mlpack::nca::NCA< MetricType, OptimizerType >::errorFunction [private]

The function to optimize.

Definition at line 111 of file nca.hpp.

21.56.4.3 template<typename MetricType = metric::SquaredEuclideanDistance, template< typename > class OptimizerType = optimization::SGD> const arma::Col<size_t>& mlpack::nca::NCA< MetricType, OptimizerType >::labels [private]

Labels reference.

Definition at line 105 of file nca.hpp.

Referenced by mlpack::nca::NCA< MetricType, OptimizerType >::Labels().

21.56.4.4 template<typename MetricType = metric::SquaredEuclideanDistance, template< typename > class OptimizerType = optimization::SGD> MetricType mlpack::nca::NCA< MetricType, OptimizerType >::metric [private]

Metric to be used.

Definition at line 108 of file nca.hpp.

21.56.4.5 template<typename MetricType = metric::SquaredEuclideanDistance, template< typename > class OptimizerType = optimization::SGD> OptimizerType<SoftmaxErrorFunction<MetricType> > mlpack::nca::NCA< MetricType, OptimizerType >::optimizer [private]

The optimizer to use.

Definition at line 114 of file nca.hpp.

Referenced by mlpack::nca::NCA< MetricType, OptimizerType >::Optimizer().

The documentation for this class was generated from the following file:

- src/mlpack/methods/nca/**nca.hpp**

21.57 mlpack::nca::SoftmaxErrorFunction< MetricType > Class Template Reference

The "softmax" stochastic neighbor assignment probability function.

Public Member Functions

- **SoftmaxErrorFunction** (const arma::mat &**dataset**, const arma::Col< size_t > &**labels**, MetricType **metric**=MetricType())

  Initialize with the given kernel; useful when the kernel has some state to store, which is set elsewhere.
- double **Evaluate** (const arma::mat &covariance)

  Evaluate the softmax function for the given covariance matrix.
- double **Evaluate** (const arma::mat &covariance, const size_t i)

  Evaluate the softmax objective function for the given covariance matrix on only one point of the dataset.

- const arma::mat **GetInitialPoint** () const

  Get the initial point.
- void **Gradient** (const arma::mat &covariance, arma::mat &gradient)

  Evaluate the gradient of the softmax function for the given covariance matrix.
- void **Gradient** (const arma::mat &covariance, const size_t i, arma::mat &gradient)

  Evaluate the gradient of the softmax function for the given covariance matrix on only one point of the dataset.
- size_t **NumFunctions** () const

  Get the number of functions the objective function can be decomposed into.

Private Member Functions

- void **Precalculate** (const arma::mat &coordinates)

  *Precalculate the denominators and numerators that will make up the p_ij, but only if the coordinates matrix is different than the last coordinates the **Precalculate()** (p. 302) method was run with.*

Private Attributes

- const arma::mat & **dataset**

  The dataset.
- arma::vec **denominators**

  *Holds denominators for calculation of p_ij, for the non-separable **Evaluate()** (p. 301) and **Gradient()** (p. 302).*
- const arma::Col< size_t > & **labels**

  Labels for each point in the dataset.
- arma::mat **lastCoordinates**

  *Last coordinates. Used for the non-separable **Evaluate()** (p. 301) and **Gradient()** (p. 302).*
- MetricType **metric**

  The instantiated metric.
- arma::vec **p**

  *Holds calculated p_i, for the non-separable **Evaluate()** (p. 301) and **Gradient()** (p. 302).*
- bool **precalculated**

  False if nothing has ever been precalculated (only at construction time).
- arma::mat **stretchedDataset**

  Stretched dataset. Kept internal to avoid memory reallocations.

21.57.1 Detailed Description

template<typename MetricType = metric::SquaredEuclideanDistance>class mlpack::nca::SoftmaxErrorFunction< MetricType >

The "softmax" stochastic neighbor assignment probability function.

The actual function is

p_ij = (exp(-|| A x_i - A x_j || ^ 2)) / (sum_{k != i} (exp(-|| A x_i - A x_k || ^ 2)))

where x_n represents a point and A is the current scaling matrix.

This class is more flexible than the original paper, allowing an arbitrary metric function to be used in place of || A x_i - A x_j ||^2, meaning that the squared Euclidean distance is not the only allowed metric for **NCA** (p. 296). However, that is probably the best way to use this class.

In addition to the standard **Evaluate()** (p. 301) and **Gradient()** (p. 302) functions which MLPACK optimizers use, overloads of **Evaluate()** (p. 301) and **Gradient()** (p. 302) are given which only operate on one point in the dataset. This is useful for optimizers like stochastic gradient descent (see **mlpack::optimization::SGD** (p. 394)).

Definition at line 52 of file nca_softmax_error_function.hpp.

21.57.2 Constructor & Destructor Documentation

21.57.2.1 template<typename MetricType = metric::SquaredEuclideanDistance> mlpack::nca::SoftmaxErrorFunction< MetricType >::SoftmaxErrorFunction (const arma::mat & *dataset*, const arma::Col< size_t > & *labels*, MetricType *metric* = `MetricType()`)

Initialize with the given kernel; useful when the kernel has some state to store, which is set elsewhere.

If no kernel is given, an empty kernel is used; this way, you can call the constructor with no arguments. A reference to the dataset we will be optimizing over is also required.

Parameters

dataset	Matrix containing the dataset.
labels	Vector of class labels for each point in the dataset.
kernel	Instantiated kernel (optional).

21.57.3 Member Function Documentation

21.57.3.1 template<typename MetricType = metric::SquaredEuclideanDistance> double mlpack::nca::SoftmaxErrorFunction< MetricType >::Evaluate (const arma::mat & *covariance*)

Evaluate the softmax function for the given covariance matrix.

This is the non-separable implementation, where the objective function is not decomposed into the sum of several objective functions.

Parameters

covariance	Covariance matrix of Mahalanobis distance.

21.57.3.2 template<typename MetricType = metric::SquaredEuclideanDistance> double mlpack::nca::SoftmaxErrorFunction< MetricType >::Evaluate (const arma::mat & *covariance*, const size_t *i*)

Evaluate the softmax objective function for the given covariance matrix on only one point of the dataset.

This is the separable implementation, where the objective function is decomposed into the sum of many objective functions, and here, only one of those constituent objective functions is returned.

Parameters

covariance	Covariance matrix of Mahalanobis distance.
i	Index of point to use for objective function.

21.57.3.3 template<typename MetricType = metric::SquaredEuclideanDistance> const arma::mat mlpack::nca::SoftmaxErrorFunction< MetricType >::GetInitialPoint ( ) const

Get the initial point.

21.57.3.4 template<typename MetricType = metric::SquaredEuclideanDistance> void mlpack::nca::SoftmaxErrorFunction< MetricType >::Gradient (const arma::mat & *covariance,* arma::mat & *gradient*)

Evaluate the gradient of the softmax function for the given covariance matrix.

This is the non-separable implementation, where the objective function is not decomposed into the sum of several objective functions.

Parameters

covariance	Covariance matrix of Mahalanobis distance.
gradient	Matrix to store the calculated gradient in.

21.57.3.5 template<typename MetricType = metric::SquaredEuclideanDistance> void mlpack::nca::SoftmaxErrorFunction< MetricType >::Gradient (const arma::mat & *covariance,* const size_t *i,* arma::mat & *gradient*)

Evaluate the gradient of the softmax function for the given covariance matrix on only one point of the dataset.

This is the separable implementation, where the objective function is decomposed into the sum of many objective functions, and here, only one of those constituent objective functions is returned.

Parameters

covariance	Covariance matrix of Mahalanobis distance.
i	Index of point to use for objective function.
gradient	Matrix to store the calculated gradient in.

21.57.3.6 template<typename MetricType = metric::SquaredEuclideanDistance> size_t mlpack::nca::SoftmaxErrorFunction< MetricType >::NumFunctions ( ) const [inline]

Get the number of functions the objective function can be decomposed into.

This is just the number of points in the dataset.

Definition at line 124 of file nca_softmax_error_function.hpp.

21.57.3.7 template<typename MetricType = metric::SquaredEuclideanDistance> void mlpack::nca::SoftmaxErrorFunction< MetricType >::Precalculate (const arma::mat & *coordinates*) [private]

Precalculate the denominators and numerators that will make up the p_ij, but only if the coordinates matrix is different than the last coordinates the **Precalculate()** (p. 302) method was run with.

This method is only called by the non-separable **Evaluate()** (p. 301) and **Gradient()** (p. 302).

This will update last_coordinates_ and stretched_dataset_, and also calculate the p_i and denominators_ which are used in the calculation of p_i or p_ij. The calculation will be O((n $*$ (n + 1)) / 2), which is not great.

Parameters

coordinates	Coordinates matrix to use for precalculation.

21.57.4 Member Data Documentation

21.57.4.1 template<typename MetricType = metric::SquaredEuclideanDistance> const arma::mat& mlpack::nca::SoftmaxErrorFunction< MetricType >::dataset [private]

The dataset.

Definition at line 128 of file nca_softmax_error_function.hpp.

21.57.4.2 template<typename MetricType = metric::SquaredEuclideanDistance> arma::vec mlpack::nca::SoftmaxErrorFunction< MetricType >::denominators [private]

Holds denominators for calculation of p_ij, for the non-separable **Evaluate()** (p. 301) and **Gradient()** (p. 302).

Definition at line 143 of file nca_softmax_error_function.hpp.

21.57.4.3 template<typename MetricType = metric::SquaredEuclideanDistance> const arma::Col<size_t>& mlpack::nca::SoftmaxErrorFunction< MetricType >::labels [private]

Labels for each point in the dataset.

Definition at line 130 of file nca_softmax_error_function.hpp.

21.57.4.4 template<typename MetricType = metric::SquaredEuclideanDistance> arma::mat mlpack::nca::SoftmaxErrorFunction< MetricType >::lastCoordinates [private]

Last coordinates. Used for the non-separable **Evaluate()** (p. 301) and **Gradient()** (p. 302).

Definition at line 136 of file nca_softmax_error_function.hpp.

21.57.4.5 template<typename MetricType = metric::SquaredEuclideanDistance> MetricType mlpack::nca::SoftmaxErrorFunction< MetricType >::metric [private]

The instantiated metric.

Definition at line 133 of file nca_softmax_error_function.hpp.

21.57.4.6 template<typename MetricType = metric::SquaredEuclideanDistance> arma::vec mlpack::nca::SoftmaxErrorFunction< MetricType >::p [private]

Holds calculated p_i, for the non-separable **Evaluate()** (p. 301) and **Gradient()** (p. 302).

Definition at line 140 of file nca_softmax_error_function.hpp.

21.57.4.7 template<typename MetricType = metric::SquaredEuclideanDistance> bool mlpack::nca::SoftmaxErrorFunction< MetricType >::precalculated [private]

False if nothing has ever been precalculated (only at construction time).

Definition at line 146 of file nca_softmax_error_function.hpp.

21.57.4.8 template<typename MetricType = metric::SquaredEuclideanDistance> arma::mat mlpack::nca::SoftmaxErrorFunction< MetricType >::stretchedDataset [private]

Stretched dataset. Kept internal to avoid memory reallocations.

Definition at line 138 of file nca_softmax_error_function.hpp.

The documentation for this class was generated from the following file:

- src/mlpack/methods/nca/**nca_softmax_error_function.hpp**

21.58 mlpack::neighbor::FurthestNeighborSort Class Reference

This class implements the necessary methods for the SortPolicy template parameter of the **NeighborSearch** (p. 317) class.

Static Public Member Functions

- static double **BestDistance** ()

  Return what should represent the best possible distance with this particular sort policy.
- template<typename TreeType >
  static double **BestNodeToNodeDistance** (const TreeType *queryNode, const TreeType *referenceNode)

  Return the best possible distance between two nodes.
- template<typename TreeType >
  static double **BestNodeToNodeDistance** (const TreeType *queryNode, const TreeType *referenceNode, const double centerToCenterDistance)

  Return the best possible distance between two nodes, given that the distance between the centers of the two nodes has already been calculated.
- template<typename TreeType >
  static double **BestNodeToNodeDistance** (const TreeType *queryNode, const TreeType *referenceNode, const TreeType *referenceChildNode, const double centerToCenterDistance)

  Return the best possible distance between the query node and the reference child node given the base case distance between the query node and the reference node.
- template<typename TreeType >
  static double **BestPointToNodeDistance** (const arma::vec &queryPoint, const TreeType *referenceNode)

  Return the best possible distance between a node and a point.
- template<typename TreeType >
  static double **BestPointToNodeDistance** (const arma::vec &queryPoint, const TreeType *referenceNode, const double pointToCenterDistance)

  Return the best possible distance between a point and a node, given that the distance between the point and the center of the node has already been calculated.
- static double **CombineBest** (const double a, const double b)

  Return the best combination of the two distances.
- static double **CombineWorst** (const double a, const double b)

  Return the worst combination of the two distances.
- static bool **IsBetter** (const double value, const double ref)

  Return whether or not value is "better" than ref.

- static size_t **SortDistance** (const arma::vec &list, double newDistance)

  Return the index in the vector where the new distance should be inserted, or size_t() - 1 if it should not be inserted (i.e.
- static double **WorstDistance** ()

  Return what should represent the worst possible distance with this particular sort policy.

21.58.1 Detailed Description

This class implements the necessary methods for the SortPolicy template parameter of the **NeighborSearch** (p. 317) class.

The sorting policy here is that the minimum distance is the best (so, when used with **NeighborSearch** (p. 317), the output is furthest neighbors).

Definition at line 37 of file furthest_neighbor_sort.hpp.

21.58.2 Member Function Documentation

21.58.2.1 static double mlpack::neighbor::FurthestNeighborSort::BestDistance ( ) [inline], [static]

Return what should represent the best possible distance with this particular sort policy.

In our case, this should be the maximum possible distance, DBL_MAX.

Returns

DBL_MAX

Definition at line 144 of file furthest_neighbor_sort.hpp.

21.58.2.2 template<typename TreeType > static double mlpack::neighbor::FurthestNeighborSort::BestNodeToNodeDistance (const TreeType * *queryNode,* const TreeType * *referenceNode*) [static]

Return the best possible distance between two nodes.

In our case, this is the maximum distance between the two tree nodes using the given distance function.

21.58.2.3 template<typename TreeType > static double mlpack::neighbor::FurthestNeighborSort::BestNodeToNodeDistance (const TreeType * *queryNode,* const TreeType * *referenceNode,* const double *centerToCenterDistance*) [static]

Return the best possible distance between two nodes, given that the distance between the centers of the two nodes has already been calculated.

This is used in conjunction with trees that have self-children (like cover trees).

21.58.2.4 template<typename TreeType > static double mlpack::neighbor::FurthestNeighborSort::BestNodeToNodeDistance (const TreeType * *queryNode,* const TreeType * *referenceNode,* const TreeType * *referenceChildNode,* const double *centerToCenterDistance*) [static]

Return the best possible distance between the query node and the reference child node given the base case distance between the query node and the reference node.

TreeType::ParentDistance() must be implemented to use this.

Parameters

queryNode	Query node.
referenceNode	Reference node.
referenceChild-Node	Child of reference node which is being inspected.
centerToCenter-Distance	Distance between centers of query node and reference node.

21.58.2.5 template<typename TreeType > static double mlpack::neighbor::FurthestNeighborSort::BestPointToNodeDistance (const arma::vec & *queryPoint,* const TreeType * *referenceNode*) [static]

Return the best possible distance between a node and a point.

In our case, this is the maximum distance between the tree node and the point using the given distance function.

21.58.2.6 template<typename TreeType > static double mlpack::neighbor::FurthestNeighborSort::BestPointToNodeDistance (const arma::vec & *queryPoint,* const TreeType * *referenceNode,* const double *pointToCenterDistance*) [static]

Return the best possible distance between a point and a node, given that the distance between the point and the center of the node has already been calculated.

This is used in conjunction with trees that have self-children (like cover trees).

21.58.2.7 static double mlpack::neighbor::FurthestNeighborSort::CombineBest (const double *a,* const double *b*) [inline], [static]

Return the best combination of the two distances.

Definition at line 149 of file furthest_neighbor_sort.hpp.

21.58.2.8 static double mlpack::neighbor::FurthestNeighborSort::CombineWorst (const double *a,* const double *b*) [inline], [static]

Return the worst combination of the two distances.

Definition at line 159 of file furthest_neighbor_sort.hpp.

21.58.2.9 static bool mlpack::neighbor::FurthestNeighborSort::IsBetter (const double *value,* const double *ref*) [inline], [static]

Return whether or not value is "better" than ref.

In this case, that means that the value is greater than the reference.

Parameters

value	Value to compare
ref	Value to compare with

Returns

bool indicating whether or not (value > ref).

Definition at line 65 of file furthest_neighbor_sort.hpp.

21.58.2.10 static size_t mlpack::neighbor::FurthestNeighborSort::SortDistance (const arma::vec & *list,* double *newDistance*) [static]

Return the index in the vector where the new distance should be inserted, or size_t() - 1 if it should not be inserted (i.e. if it is not any better than any of the existing points in the list). The list should be sorted such that the best point is the first in the list. The actual insertion is not performed.

Parameters

list	Vector of existing distance points, sorted such that the best point is the first in the list.
new_distance	Distance to try to insert.

Returns

size_t containing the position to insert into, or (size_t() - 1) if the new distance should not be inserted.

21.58.2.11 static double mlpack::neighbor::FurthestNeighborSort::WorstDistance ( ) [inline],[static]

Return what should represent the worst possible distance with this particular sort policy.

In our case, this should be the minimum possible distance, 0.

Returns

0

Definition at line 135 of file furthest_neighbor_sort.hpp.

The documentation for this class was generated from the following file:

- src/mlpack/methods/neighbor_search/sort_policies/**furthest_neighbor_sort.hpp**

21.59 mlpack::neighbor::LSHSearch< SortPolicy > Class Template Reference

The **LSHSearch** (p. 307) class – This class builds a hash on the reference set and uses this hash to compute the distance-approximate nearest-neighbors of the given queries.

Public Member Functions

- **LSHSearch** (const arma::mat &**referenceSet**, const arma::mat &**querySet**, const size_t **numProj**, const size_t **numTables**, const double **hashWidth**=0.0, const size_t **secondHashSize**=99901, const size_t **bucketSize**=500)

  This function initializes the LSH class.
- **LSHSearch** (const arma::mat &**referenceSet**, const size_t **numProj**, const size_t **numTables**, const double **hashWidth**=0.0, const size_t **secondHashSize**=99901, const size_t **bucketSize**=500)

  This function initializes the LSH class.

- void **Search** (const size_t k, arma::Mat< size_t > &resultingNeighbors, arma::mat &distances, const size_t numTablesToSearch=0)

  Compute the nearest neighbors and store the output in the given matrices.

Private Member Functions

- double **BaseCase** (const size_t queryIndex, const size_t referenceIndex)

  This is a helper function that computes the distance of the query to the neighbor candidates and appropriately stores the best 'k' candidates.
- void **BuildHash** ()

  This function builds a hash table with two levels of hashing as presented in the paper.
- void **InsertNeighbor** (const size_t queryIndex, const size_t pos, const size_t neighbor, const double distance)

  This is a helper function that efficiently inserts better neighbor candidates into an existing set of neighbor candidates.
- void **ReturnIndicesFromTable** (const size_t queryIndex, arma::uvec &referenceIndices, size_t numTablesToSearch)

  This function takes a query and hashes it into each of the hash tables to get keys for the query and then the key is hashed to a bucket of the second hash table and all the points (if any) in those buckets are collected as the potential neighbor candidates.

Private Attributes

- arma::Col< size_t > **bucketContentSize**

  The number of elements present in each hash bucket; should be secondHashSize.
- arma::Col< size_t > **bucketRowInHashTable**

  For a particular hash value, points to the row in secondHashTable corresponding to this value.
- const size_t **bucketSize**

  The bucket size of the second hash.
- arma::mat * **distancePtr**

  The pointer to the nearest neighbor distances.
- double **hashWidth**

  The hash width.
- **metric::SquaredEuclideanDistance metric**

  Instantiation of the metric.
- arma::Mat< size_t > * **neighborPtr**

  The pointer to the nearest neighbor indices.
- const size_t **numProj**

  The number of projections.
- const size_t **numTables**

  The number of hash tables.
- arma::mat **offsets**

  The list of the offset 'b' for each of the projection for each table.
- std::vector< arma::mat > **projections**

  The std::vector containing the projection matrix of each table.
- const arma::mat & **querySet**

  Query dataset (may not be given).
- const arma::mat & **referenceSet**

  Reference dataset.

- const size_t **secondHashSize**

  The big prime representing the size of the second hash.
- arma::Mat< size_t > **secondHashTable**

  The final hash table; should be (< secondHashSize) x bucketSize.
- arma::vec **secondHashWeights**

  The weights of the second hash.

21.59.1 Detailed Description

template<typename SortPolicy = NearestNeighborSort>class mlpack::neighbor::LSHSearch< SortPolicy >

The **LSHSearch** (p. 307) class – This class builds a hash on the reference set and uses this hash to compute the distance-approximate nearest-neighbors of the given queries.

Template Parameters

SortPolicy	The sort policy for distances; see **NearestNeighborSort** (p. 314).

Definition at line 59 of file lsh_search.hpp.

21.59.2 Constructor & Destructor Documentation

21.59.2.1 template<typename SortPolicy = NearestNeighborSort> mlpack::neighbor::LSHSearch< SortPolicy >::LSHSearch (const arma::mat & *referenceSet,* const arma::mat & *querySet,* const size_t *numProj,* const size_t *numTables,* const double *hashWidth* = 0.0, const size_t *secondHashSize* = 99901, const size_t *bucketSize* = 500)

This function initializes the LSH class.

It builds the hash on the reference set with 2-stable distributions. See the individual functions performing the hashing for details on how the hashing is done.

Parameters

referenceSet	Set of reference points.
querySet	Set of query points.
numProj	Number of projections in each hash table (anything between 10-50 might be a decent choice).
numTables	Total number of hash tables (anything between 10-20 should suffice).
hashWidth	The width of hash for every table. If 0 (the default) is provided, then the hash width is automatically obtained by computing the average pairwise distance of 25 pairs. This should be a reasonable upper bound on the nearest-neighbor distance in general.
secondHashSize	The size of the second hash table. This should be a large prime number.
bucketSize	The size of the bucket in the second hash table. This is the maximum number of points that can be hashed into single bucket. Default values are already provided here.

21.59.2.2 template<typename SortPolicy = NearestNeighborSort> mlpack::neighbor::LSHSearch< SortPolicy >::LSHSearch (const arma::mat & *referenceSet,* const size_t *numProj,* const size_t *numTables,* const double *hashWidth* = 0.0, const size_t *secondHashSize* = 99901, const size_t *bucketSize* = 500)

This function initializes the LSH class.

It builds the hash on the reference set with 2-stable distributions. See the individual functions performing the hashing for details on how the hashing is done.

Parameters

referenceSet	Set of reference points and the set of queries.
numProj	Number of projections in each hash table (anything between 10-50 might be a decent choice).
numTables	Total number of hash tables (anything between 10-20 should suffice).
hashWidth	The width of hash for every table. If 0 (the default) is provided, then the hash width is automatically obtained by computing the average pairwise distance of 25 pairs. This should be a reasonable upper bound on the nearest-neighbor distance in general.
secondHashSize	The size of the second hash table. This should be a large prime number.
bucketSize	The size of the bucket in the second hash table. This is the maximum number of points that can be hashed into single bucket. Default values are already provided here.

21.59.3 Member Function Documentation

21.59.3.1 template<typename SortPolicy = NearestNeighborSort> double mlpack::neighbor::LSHSearch< SortPolicy >::BaseCase (const size_t *queryIndex*, const size_t *referenceIndex*) [private]

This is a helper function that computes the distance of the query to the neighbor candidates and appropriately stores the best 'k' candidates.

Parameters

queryIndex	The index of the query in question
referenceIndex	The index of the neighbor candidate in question

21.59.3.2 template<typename SortPolicy = NearestNeighborSort> void mlpack::neighbor::LSHSearch< SortPolicy >::BuildHash ( ) [private]

This function builds a hash table with two levels of hashing as presented in the paper.

This function first hashes the points with 'numProj' random projections to a single hash table creating (key, point ID) pairs where the key is a 'numProj'-dimensional integer vector.

Then each key in this hash table is hashed into a second hash table using a standard hash.

This function does not have any parameters and relies on parameters which are private members of this class, intialized during the class intialization.

21.59.3.3 template<typename SortPolicy = NearestNeighborSort> void mlpack::neighbor::LSHSearch< SortPolicy >::InsertNeighbor (const size_t *queryIndex*, const size_t *pos*, const size_t *neighbor*, const double *distance*) [private]

This is a helper function that efficiently inserts better neighbor candidates into an existing set of neighbor candidates.

This function is only called by the 'BaseCase' function.

Parameters

queryIndex	This is the index of the query being processed currently
pos	The position of the neighbor candidate in the current list of neighbor candidates.
neighbor	The neighbor candidate that is being inserted into the list of the best 'k' candidates for the query in question.

distance	The distance of the query to the neighbor candidate.

21.59.3.4 template<typename SortPolicy = NearestNeighborSort> void mlpack::neighbor::LSHSearch< SortPolicy >::ReturnIndicesFromTable (const size_t *queryIndex*, arma::uvec & *referenceIndices*, size_t *numTablesToSearch*) [private]

This function takes a query and hashes it into each of the hash tables to get keys for the query and then the key is hashed to a bucket of the second hash table and all the points (if any) in those buckets are collected as the potential neighbor candidates.

Parameters

queryIndex	The index of the query currently being processed.
referenceIndices	The list of neighbor candidates obtained from hashing the query into all the hash tables and eventually into multiple buckets of the second hash table.

21.59.3.5 template<typename SortPolicy = NearestNeighborSort> void mlpack::neighbor::LSHSearch< SortPolicy >::Search (const size_t *k*, arma::Mat< size_t > & *resultingNeighbors*, arma::mat & *distances*, const size_t *numTablesToSearch* = 0)

Compute the nearest neighbors and store the output in the given matrices.

The matrices will be set to the size of n columns by k rows, where n is the number of points in the query dataset and k is the number of neighbors being searched for.

Parameters

k	Number of neighbors to search for.
resulting-Neighbors	Matrix storing lists of neighbors for each query point.
distances	Matrix storing distances of neighbors for each query point.
numTablesTo-Search	This parameter allows the user to have control over the number of hash tables to be searched. This allows the user to pick the number of tables it can afford for the time available without having to build hashing for every table size. By default, this is set to zero in which case all tables are considered.

21.59.4 Member Data Documentation

21.59.4.1 template<typename SortPolicy = NearestNeighborSort> arma::Col<size_t> mlpack::neighbor::LSHSearch< SortPolicy >::bucketContentSize [private]

The number of elements present in each hash bucket; should be secondHashSize.

Definition at line 235 of file lsh_search.hpp.

21.59.4.2 template<typename SortPolicy = NearestNeighborSort> arma::Col<size_t> mlpack::neighbor::LSHSearch< SortPolicy >::bucketRowInHashTable [private]

For a particular hash value, points to the row in secondHashTable corresponding to this value.

Should be secondHashSize.

Definition at line 239 of file lsh_search.hpp.

21.59.4.3 template<typename SortPolicy = NearestNeighborSort> const size_t mlpack::neighbor::LSHSearch< SortPolicy >::bucketSize [private]

The bucket size of the second hash.

Definition at line 225 of file lsh_search.hpp.

21.59.4.4 template<typename SortPolicy = NearestNeighborSort> arma::mat* mlpack::neighbor::LSHSearch< SortPolicy >::distancePtr [private]

The pointer to the nearest neighbor distances.

Definition at line 242 of file lsh_search.hpp.

21.59.4.5 template<typename SortPolicy = NearestNeighborSort> double mlpack::neighbor::LSHSearch< SortPolicy >::hashWidth [private]

The hash width.

Definition at line 216 of file lsh_search.hpp.

21.59.4.6 template<typename SortPolicy = NearestNeighborSort> metric::SquaredEuclideanDistance mlpack::neighbor::LSHSearch< SortPolicy >::metric [private]

Instantiation of the metric.

Definition at line 228 of file lsh_search.hpp.

21.59.4.7 template<typename SortPolicy = NearestNeighborSort> arma::Mat<size_t>* mlpack::neighbor::LSHSearch< SortPolicy >::neighborPtr [private]

The pointer to the nearest neighbor indices.

Definition at line 245 of file lsh_search.hpp.

21.59.4.8 template<typename SortPolicy = NearestNeighborSort> const size_t mlpack::neighbor::LSHSearch< SortPolicy >::numProj [private]

The number of projections.

Definition at line 204 of file lsh_search.hpp.

21.59.4.9 template<typename SortPolicy = NearestNeighborSort> const size_t mlpack::neighbor::LSHSearch< SortPolicy >::numTables [private]

The number of hash tables.

Definition at line 207 of file lsh_search.hpp.

21.59.4.10 template<typename SortPolicy = NearestNeighborSort> arma::mat mlpack::neighbor::LSHSearch< SortPolicy >::offsets [private]

The list of the offset 'b' for each of the projection for each table.

Definition at line 213 of file lsh_search.hpp.

21.59.4.11 template<typename SortPolicy = NearestNeighborSort> std::vector<arma::mat> mlpack::neighbor::LSHSearch< SortPolicy >::projections [private]

The std::vector containing the projection matrix of each table.

Definition at line 210 of file lsh_search.hpp.

21.59.4.12 template<typename SortPolicy = NearestNeighborSort> const arma::mat& mlpack::neighbor::LSHSearch< SortPolicy >::querySet [private]

Query dataset (may not be given).

Definition at line 201 of file lsh_search.hpp.

21.59.4.13 template<typename SortPolicy = NearestNeighborSort> const arma::mat& mlpack::neighbor::LSHSearch< SortPolicy >::referenceSet [private]

Reference dataset.

Definition at line 198 of file lsh_search.hpp.

21.59.4.14 template<typename SortPolicy = NearestNeighborSort> const size_t mlpack::neighbor::LSHSearch< SortPolicy >::secondHashSize [private]

The big prime representing the size of the second hash.

Definition at line 219 of file lsh_search.hpp.

21.59.4.15 template<typename SortPolicy = NearestNeighborSort> arma::Mat<size_t> mlpack::neighbor::LSHSearch< SortPolicy >::secondHashTable [private]

The final hash table; should be (< secondHashSize) x bucketSize.

Definition at line 231 of file lsh_search.hpp.

21.59.4.16 template<typename SortPolicy = NearestNeighborSort> arma::vec mlpack::neighbor::LSHSearch< SortPolicy >::secondHashWeights [private]

The weights of the second hash.

Definition at line 222 of file lsh_search.hpp.

The documentation for this class was generated from the following file:

- src/mlpack/methods/lsh/**lsh_search.hpp**

21.60 mlpack::neighbor::NearestNeighborSort Class Reference

This class implements the necessary methods for the SortPolicy template parameter of the **NeighborSearch** (p. 317) class.

Static Public Member Functions

- static double **BestDistance** ()

  Return what should represent the best possible distance with this particular sort policy.
- template<typename TreeType >
  static double **BestNodeToNodeDistance** (const TreeType *queryNode, const TreeType *referenceNode)

  Return the best possible distance between two nodes.
- template<typename TreeType >
  static double **BestNodeToNodeDistance** (const TreeType *queryNode, const TreeType *referenceNode, const double centerToCenterDistance)

  Return the best possible distance between two nodes, given that the distance between the centers of the two nodes has already been calculated.
- template<typename TreeType >
  static double **BestNodeToNodeDistance** (const TreeType *queryNode, const TreeType *referenceNode, const TreeType *referenceChildNode, const double centerToCenterDistance)

  Return the best possible distance between the query node and the reference child node given the base case distance between the query node and the reference node.
- template<typename TreeType >
  static double **BestPointToNodeDistance** (const arma::vec &queryPoint, const TreeType *referenceNode)

  Return the best possible distance between a node and a point.
- template<typename TreeType >
  static double **BestPointToNodeDistance** (const arma::vec &queryPoint, const TreeType *referenceNode, const double pointToCenterDistance)

  Return the best possible distance between a point and a node, given that the distance between the point and the center of the node has already been calculated.
- static double **CombineBest** (const double a, const double b)

  Return the best combination of the two distances.
- static double **CombineWorst** (const double a, const double b)

  Return the worst combination of the two distances.
- static bool **IsBetter** (const double value, const double ref)

  Return whether or not value is "better" than ref.
- static size_t **SortDistance** (const arma::vec &list, double newDistance)

  Return the index in the vector where the new distance should be inserted, or (size_t() - 1) if it should not be inserted (i.e.
- static double **WorstDistance** ()

  Return what should represent the worst possible distance with this particular sort policy.

21.60.1 Detailed Description

This class implements the necessary methods for the SortPolicy template parameter of the **NeighborSearch** (p. 317) class.

The sorting policy here is that the minimum distance is the best (so, when used with **NeighborSearch** (p. 317), the output is nearest neighbors).

This class is also meant to serve as a guide to implement a custom SortPolicy. All of the methods implemented here must be implemented by any other SortPolicy classes.

Definition at line 41 of file nearest_neighbor_sort.hpp.

21.60.2 Member Function Documentation

21.60.2.1 static double mlpack::neighbor::NearestNeighborSort::BestDistance ( ) [inline],[static]

Return what should represent the best possible distance with this particular sort policy.

In our case, this should be the minimum possible distance, 0.0.

Returns

0.0

Definition at line 147 of file nearest_neighbor_sort.hpp.

21.60.2.2 template<typename TreeType > static double mlpack::neighbor::NearestNeighborSort::BestNodeToNodeDistance (const TreeType * *queryNode,* const TreeType * *referenceNode*) [static]

Return the best possible distance between two nodes.

In our case, this is the minimum distance between the two tree nodes using the given distance function.

21.60.2.3 template<typename TreeType > static double mlpack::neighbor::NearestNeighborSort::BestNodeToNodeDistance (const TreeType * *queryNode,* const TreeType * *referenceNode,* const double *centerToCenterDistance*) [static]

Return the best possible distance between two nodes, given that the distance between the centers of the two nodes has already been calculated.

This is used in conjunction with trees that have self-children (like cover trees).

21.60.2.4 template<typename TreeType > static double mlpack::neighbor::NearestNeighborSort::BestNodeToNodeDistance (const TreeType * *queryNode,* const TreeType * *referenceNode,* const TreeType * *referenceChildNode,* const double *centerToCenterDistance*) [static]

Return the best possible distance between the query node and the reference child node given the base case distance between the query node and the reference node.

TreeType::ParentDistance() must be implemented to use this.

Parameters

queryNode	Query node.
referenceNode	Reference node.
referenceChild-Node	Child of reference node which is being inspected.

centerToCenter-Distance	Distance between centers of query node and reference node.

21.60.2.5 template<typename TreeType > static double mlpack::neighbor::NearestNeighborSort::BestPointToNodeDistance (const arma::vec & *queryPoint,* const TreeType * *referenceNode*) [static]

Return the best possible distance between a node and a point.

In our case, this is the minimum distance between the tree node and the point using the given distance function.

21.60.2.6 template<typename TreeType > static double mlpack::neighbor::NearestNeighborSort::BestPointToNodeDistance (const arma::vec & *queryPoint,* const TreeType * *referenceNode,* const double *pointToCenterDistance*) [static]

Return the best possible distance between a point and a node, given that the distance between the point and the center of the node has already been calculated.

This is used in conjunction with trees that have self-children (like cover trees).

21.60.2.7 static double mlpack::neighbor::NearestNeighborSort::CombineBest (const double *a,* const double *b*) [inline], [static]

Return the best combination of the two distances.

Definition at line 152 of file nearest_neighbor_sort.hpp.

21.60.2.8 static double mlpack::neighbor::NearestNeighborSort::CombineWorst (const double *a,* const double *b*) [inline], [static]

Return the worst combination of the two distances.

Definition at line 160 of file nearest_neighbor_sort.hpp.

21.60.2.9 static bool mlpack::neighbor::NearestNeighborSort::IsBetter (const double *value,* const double *ref*) [inline], [static]

Return whether or not value is "better" than ref.

In this case, that means that the value is less than the reference.

Parameters

value	Value to compare
ref	Value to compare with

Returns

bool indicating whether or not (value $<$ ref).

Definition at line 69 of file nearest_neighbor_sort.hpp.

21.60.2.10 static size_t mlpack::neighbor::NearestNeighborSort::SortDistance (const arma::vec & *list,* double *newDistance*) [static]

Return the index in the vector where the new distance should be inserted, or (size_t() - 1) if it should not be inserted (i.e.

if it is not any better than any of the existing points in the list). The list should be sorted such that the best point is the first in the list. The actual insertion is not performed.

Parameters

list	Vector of existing distance points, sorted such that the best point is first in the list.
new_distance	Distance to try to insert

Returns

size_t containing the position to insert into, or (size_t() - 1) if the new distance should not be inserted.

21.60.2.11 static double mlpack::neighbor::NearestNeighborSort::WorstDistance ( ) [inline], [static]

Return what should represent the worst possible distance with this particular sort policy.

In our case, this should be the maximum possible distance, DBL_MAX.

Returns

DBL_MAX

Definition at line 138 of file nearest_neighbor_sort.hpp.

The documentation for this class was generated from the following file:

- src/mlpack/methods/neighbor_search/sort_policies/**nearest_neighbor_sort.hpp**

21.61 mlpack::neighbor::NeighborSearch< SortPolicy, MetricType, TreeType > Class Template Reference

The **NeighborSearch** (p. 317) class is a template class for performing distance-based neighbor searches.

Public Member Functions

- **NeighborSearch** (const typename TreeType::Mat &**referenceSet**, const typename TreeType::Mat &**querySet**, const bool **naive**=false, const bool **singleMode**=false, const size_t leafSize=20, const MetricType **metric**=MetricType())

  *Initialize the **NeighborSearch** (p. 317) object, passing both a query and reference dataset.*
- **NeighborSearch** (const typename TreeType::Mat &**referenceSet**, const bool **naive**=false, const bool **singleMode**=false, const size_t leafSize=20, const MetricType **metric**=MetricType())

  *Initialize the **NeighborSearch** (p. 317) object, passing only one dataset, which is used as both the query and the reference dataset.*
- **NeighborSearch** (TreeType ***__referenceTree__, TreeType ***__queryTree__, const typename TreeType::Mat &**referenceSet**, const typename TreeType::Mat &**querySet**, const bool **singleMode**=false, const MetricType **metric**=MetricType())

*Initialize the **NeighborSearch** (p. 317) object with the given datasets and pre-constructed trees.*

- **NeighborSearch** (TreeType ∗**referenceTree**, const typename TreeType::Mat &**referenceSet**, const bool **singleMode**=false, const MetricType **metric**=MetricType())

  *Initialize the **NeighborSearch** (p. 317) object with the given reference dataset and pre-constructed tree.*
- ∼**NeighborSearch** ()

  *Delete the **NeighborSearch** (p. 317) object.*
- void **Search** (const size_t k, arma::Mat< size_t > &resultingNeighbors, arma::mat &distances)

  Compute the nearest neighbors and store the output in the given matrices.

Private Attributes

- bool **hasQuerySet**

  Indicates if a separate query set was passed.
- MetricType **metric**

  Instantiation of metric.
- bool **naive**

  Indicates if O(n^2) naive search is being used.
- size_t **numberOfPrunes**

  Total number of pruned nodes during the neighbor search.
- std::vector< size_t > **oldFromNewQueries**

  Permutations of query points during tree building.
- std::vector< size_t > **oldFromNewReferences**

  Permutations of reference points during tree building.
- arma::mat **queryCopy**

  Copy of query dataset (if we need it, because tree building modifies it).
- const arma::mat & **querySet**

  Query dataset (may not be given).
- TreeType ∗ **queryTree**

  Pointer to the root of the query tree (might not exist).
- arma::mat **referenceCopy**

  Copy of reference dataset (if we need it, because tree building modifies it).
- const arma::mat & **referenceSet**

  Reference dataset.
- TreeType ∗ **referenceTree**

  Pointer to the root of the reference tree.
- bool **singleMode**

  Indicates if single-tree search is being used (opposed to dual-tree).
- bool **treeOwner**

  If true, this object created the trees and is responsible for them.

21.61.1 Detailed Description

template<typename SortPolicy = NearestNeighborSort, typename MetricType = mlpack::metric::SquaredEuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, NeighborSearchStat<SortPolicy> >>class mlpack::neighbor::NeighborSearch< SortPolicy, MetricType, TreeType >

The **NeighborSearch** (p. 317) class is a template class for performing distance-based neighbor searches.

It takes a query dataset and a reference dataset (or just a reference dataset) and, for each point in the query dataset, finds the k neighbors in the reference dataset which have the 'best' distance according to a given sorting policy. A constructor is given which takes only a reference dataset, and if that constructor is used, the given reference dataset is also used as the query dataset.

The template parameters SortPolicy and Metric define the sort function used and the metric (distance function) used. More information on those classes can be found in the **NearestNeighborSort** (p. 314) class and the **kernel::ExampleKernel** (p. 232) class.

Template Parameters

SortPolicy	The sort policy for distances; see **NearestNeighborSort** (p. 314).
MetricType	The metric to use for computation.
TreeType	The tree type to use.

Definition at line 63 of file neighbor_search.hpp.

21.61.2 Constructor & Destructor Documentation

21.61.2.1 template<typename SortPolicy = NearestNeighborSort, typename MetricType = mlpack::metric::SquaredEuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, NeighborSearchStat<SortPolicy> >> mlpack::neighbor::NeighborSearch< SortPolicy, MetricType, TreeType >::NeighborSearch (const typename TreeType::Mat & *referenceSet*, const typename TreeType::Mat & *querySet*, const bool *naive* = false, const bool *singleMode* = false, const size_t *leafSize* = 20, const MetricType *metric* = MetricType())

Initialize the **NeighborSearch** (p. 317) object, passing both a query and reference dataset.

Optionally, perform the computation in naive mode or single-tree mode, and set the leaf size used for tree-building. An initialized distance metric can be given, for cases where the metric has internal data (i.e. the distance::MahalanobisDistance class).

This method will copy the matrices to internal copies, which are rearranged during tree-building. You can avoid this extra copy by pre-constructing the trees and passing them using a diferent constructor.

Parameters

referenceSet	Set of reference points.
querySet	Set of query points.
naive	If true, O(n^2) naive search will be used (as opposed to dual-tree search). This overrides singleMode (if it is set to true).
singleMode	If true, single-tree search will be used (as opposed to dual-tree search).
leafSize	Leaf size for tree construction (ignored if tree is given).
metric	An optional instance of the MetricType class.

21.61.2.2 template<typename SortPolicy = NearestNeighborSort, typename MetricType = mlpack::metric::SquaredEuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, NeighborSearchStat<SortPolicy> >> mlpack::neighbor::NeighborSearch< SortPolicy, MetricType, TreeType >::NeighborSearch (const typename TreeType::Mat & *referenceSet*, const bool *naive* = false, const bool *singleMode* = false, const size_t *leafSize* = 20, const MetricType *metric* = MetricType())

Initialize the **NeighborSearch** (p. 317) object, passing only one dataset, which is used as both the query and the reference dataset.

Optionally, perform the computation in naive mode or single-tree mode, and set the leaf size used for tree-building. An initialized distance metric can be given, for cases where the metric has internal data (i.e. the distance::MahalanobisDistance class).

If naive mode is being used and a pre-built tree is given, it may not work: naive mode operates by building a one-node tree (the root node holds all the points). If that condition is not satisfied with the pre-built tree, then naive mode will not work.

Parameters

referenceSet	Set of reference points.
naive	If true, O(n^2) naive search will be used (as opposed to dual-tree search). This overrides singleMode (if it is set to true).
singleMode	If true, single-tree search will be used (as opposed to dual-tree search).
leafSize	Leaf size for tree construction (ignored if tree is given).
metric	An optional instance of the MetricType class.

21.61.2.3 template<typename SortPolicy = NearestNeighborSort, typename MetricType = mlpack::metric::SquaredEuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, NeighborSearchStat<SortPolicy> >> mlpack::neighbor::NeighborSearch< SortPolicy, MetricType, TreeType >::NeighborSearch (TreeType * *referenceTree,* TreeType * *queryTree,* const typename TreeType::Mat & *referenceSet,* const typename TreeType::Mat & *querySet,* const bool *singleMode* = `false`, const MetricType *metric* = `MetricType()`)

Initialize the **NeighborSearch** (p. 317) object with the given datasets and pre-constructed trees.

It is assumed that the points in referenceSet and querySet correspond to the points in referenceTree and queryTree, respectively. Optionally, choose to use single-tree mode. Naive mode is not available as an option for this constructor; instead, to run naive computation, construct a tree with all of the points in one leaf (i.e. leafSize = number of points). Additionally, an instantiated distance metric can be given, for cases where the distance metric holds data.

There is no copying of the data matrices in this constructor (because tree-building is not necessary), so this is the constructor to use when copies absolutely must be avoided.

Note

Because tree-building (at least with BinarySpaceTree) modifies the ordering of a matrix, be sure you pass the modified matrix to this object! In addition, mapping the points of the matrix back to their original indices is not done when this constructor is used.

Parameters

referenceTree	Pre-built tree for reference points.
queryTree	Pre-built tree for query points.
referenceSet	Set of reference points corresponding to referenceTree.
querySet	Set of query points corresponding to queryTree.
singleMode	Whether single-tree computation should be used (as opposed to dual-tree computation).
metric	Instantiated distance metric.

21.61.2.4 template<typename SortPolicy = NearestNeighborSort, typename MetricType = mlpack::metric::SquaredEuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, NeighborSearchStat<SortPolicy> >> mlpack::neighbor::NeighborSearch< SortPolicy, MetricType, TreeType >::NeighborSearch (TreeType * *referenceTree,* const typename TreeType::Mat & *referenceSet,* const bool *singleMode* = `false`, const MetricType *metric* = `MetricType()`)

Initialize the **NeighborSearch** (p. 317) object with the given reference dataset and pre-constructed tree.

It is assumed that the points in referenceSet correspond to the points in referenceTree. Optionally, choose to use single-tree mode. Naive mode is not available as an option for this constructor; instead, to run naive computation, construct a

tree with all the points in one leaf (i.e. leafSize = number of points). Additionally, an instantiated distance metric can be given, for the case where the distance metric holds data.

There is no copying of the data matrices in this constructor (because tree-building is not necessary), so this is the constructor to use when copies absolutely must be avoided.

Note

Because tree-building (at least with BinarySpaceTree) modifies the ordering of a matrix, be sure you pass the modified matrix to this object! In addition, mapping the points of the matrix back to their original indices is not done when this constructor is used.

Parameters

referenceTree	Pre-built tree for reference points.
referenceSet	Set of reference points corresponding to referenceTree.
singleMode	Whether single-tree computation should be used (as opposed to dual-tree computation).
metric	Instantiated distance metric.

21.61.2.5 template<typename SortPolicy = NearestNeighborSort, typename MetricType = mlpack::metric::SquaredEuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, NeighborSearchStat<SortPolicy> >> mlpack::neighbor::NeighborSearch< SortPolicy, MetricType, TreeType >::~NeighborSearch ( )

Delete the **NeighborSearch** (p. 317) object.

The tree is the only member we are responsible for deleting. The others will take care of themselves.

21.61.3 Member Function Documentation

21.61.3.1 template<typename SortPolicy = NearestNeighborSort, typename MetricType = mlpack::metric::SquaredEuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, NeighborSearchStat<SortPolicy> >> void mlpack::neighbor::NeighborSearch< SortPolicy, MetricType, TreeType >::Search (const size_t *k*, arma::Mat< size_t > & *resultingNeighbors*, arma::mat & *distances*)

Compute the nearest neighbors and store the output in the given matrices.

The matrices will be set to the size of n columns by k rows, where n is the number of points in the query dataset and k is the number of neighbors being searched for.

Parameters

k	Number of neighbors to search for.
resulting-Neighbors	Matrix storing lists of neighbors for each query point.
distances	Matrix storing distances of neighbors for each query point.

21.61.4 Member Data Documentation

21.61.4.1 template<typename SortPolicy = NearestNeighborSort, typename MetricType = mlpack::metric::SquaredEuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, NeighborSearchStat<SortPolicy> >> bool mlpack::neighbor::NeighborSearch< SortPolicy, MetricType, TreeType >::hasQuerySet [private]

Indicates if a separate query set was passed.

Definition at line 231 of file neighbor_search.hpp.

21.61.4.2 template<typename SortPolicy = NearestNeighborSort, typename MetricType = mlpack::metric::SquaredEuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, NeighborSearchStat<SortPolicy> >> MetricType mlpack::neighbor::NeighborSearch< SortPolicy, MetricType, TreeType >::metric [private]

Instantiation of metric.

Definition at line 239 of file neighbor_search.hpp.

21.61.4.3 template<typename SortPolicy = NearestNeighborSort, typename MetricType = mlpack::metric::SquaredEuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, NeighborSearchStat<SortPolicy> >> bool mlpack::neighbor::NeighborSearch< SortPolicy, MetricType, TreeType >::naive [private]

Indicates if O(n^2) naive search is being used.

Definition at line 234 of file neighbor_search.hpp.

21.61.4.4 template<typename SortPolicy = NearestNeighborSort, typename MetricType = mlpack::metric::SquaredEuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, NeighborSearchStat<SortPolicy> >> size_t mlpack::neighbor::NeighborSearch< SortPolicy, MetricType, TreeType >::numberOfPrunes [private]

Total number of pruned nodes during the neighbor search.

Definition at line 247 of file neighbor_search.hpp.

21.61.4.5 template<typename SortPolicy = NearestNeighborSort, typename MetricType = mlpack::metric::SquaredEuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, NeighborSearchStat<SortPolicy> >> std::vector<size_t> mlpack::neighbor::NeighborSearch< SortPolicy, MetricType, TreeType >::oldFromNewQueries [private]

Permutations of query points during tree building.

Definition at line 244 of file neighbor_search.hpp.

21.61.4.6 template<typename SortPolicy = NearestNeighborSort, typename MetricType = mlpack::metric::SquaredEuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, NeighborSearchStat<SortPolicy> >> std::vector<size_t> mlpack::neighbor::NeighborSearch< SortPolicy, MetricType, TreeType >::oldFromNewReferences [private]

Permutations of reference points during tree building.

Definition at line 242 of file neighbor_search.hpp.

21.61.4.7 template<typename SortPolicy = NearestNeighborSort, typename MetricType = mlpack::metric::SquaredEuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, NeighborSearchStat<SortPolicy> >> arma::mat mlpack::neighbor::NeighborSearch< SortPolicy, MetricType, TreeType >::queryCopy [private]

Copy of query dataset (if we need it, because tree building modifies it).

Definition at line 216 of file neighbor_search.hpp.

21.61.4.8 template<typename SortPolicy = NearestNeighborSort, typename MetricType = mlpack::metric::SquaredEuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, NeighborSearchStat<SortPolicy> >> const arma::mat& mlpack::neighbor::NeighborSearch< SortPolicy, MetricType, TreeType >::querySet [private]

Query dataset (may not be given).

Definition at line 221 of file neighbor_search.hpp.

21.61.4.9 template<typename SortPolicy = NearestNeighborSort, typename MetricType = mlpack::metric::SquaredEuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, NeighborSearchStat<SortPolicy> >> TreeType* mlpack::neighbor::NeighborSearch< SortPolicy, MetricType, TreeType >::queryTree [private]

Pointer to the root of the query tree (might not exist).

Definition at line 226 of file neighbor_search.hpp.

21.61.4.10 template<typename SortPolicy = NearestNeighborSort, typename MetricType = mlpack::metric::SquaredEuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, NeighborSearchStat<SortPolicy> >> arma::mat mlpack::neighbor::NeighborSearch< SortPolicy, MetricType, TreeType >::referenceCopy [private]

Copy of reference dataset (if we need it, because tree building modifies it).

Definition at line 214 of file neighbor_search.hpp.

21.61.4.11 template<typename SortPolicy = NearestNeighborSort, typename MetricType = mlpack::metric::SquaredEuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, NeighborSearchStat<SortPolicy> >> const arma::mat& mlpack::neighbor::NeighborSearch< SortPolicy, MetricType, TreeType >::referenceSet [private]

Reference dataset.

Definition at line 219 of file neighbor_search.hpp.

21.61.4.12 template<typename SortPolicy = NearestNeighborSort, typename MetricType = mlpack::metric::SquaredEuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, NeighborSearchStat<SortPolicy> >> TreeType* mlpack::neighbor::NeighborSearch< SortPolicy, MetricType, TreeType >::referenceTree [private]

Pointer to the root of the reference tree.

Definition at line 224 of file neighbor_search.hpp.

21.61.4.13 template<typename SortPolicy = NearestNeighborSort, typename MetricType = mlpack::metric::SquaredEuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, NeighborSearchStat<SortPolicy> >> bool mlpack::neighbor::NeighborSearch< SortPolicy, MetricType, TreeType >::singleMode [private]

Indicates if single-tree search is being used (opposed to dual-tree).

Definition at line 236 of file neighbor_search.hpp.

21.61.4.14 template<typename SortPolicy = NearestNeighborSort, typename MetricType = mlpack::metric::SquaredEuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, NeighborSearchStat<SortPolicy> >> bool mlpack::neighbor::NeighborSearch< SortPolicy, MetricType, TreeType >::treeOwner [private]

If true, this object created the trees and is responsible for them.

Definition at line 229 of file neighbor_search.hpp.

The documentation for this class was generated from the following file:

- src/mlpack/methods/neighbor_search/**neighbor_search.hpp**

21.62 mlpack::neighbor::NeighborSearchRules< SortPolicy, MetricType, TreeType > Class Template Reference

Public Member Functions

- **NeighborSearchRules** (const arma::mat &**referenceSet**, const arma::mat &**querySet**, arma::Mat< size_t > &**neighbors**, arma::mat &**distances**, MetricType &**metric**)
- double **BaseCase** (const size_t queryIndex, const size_t referenceIndex)
- double **Rescore** (const size_t queryIndex, TreeType &referenceNode, const double oldScore) const

  Re-evaluate the score for recursion order.
- double **Rescore** (TreeType &queryNode, TreeType &referenceNode, const double oldScore) const

  Re-evaluate the score for recursion order.
- double **Score** (const size_t queryIndex, TreeType &referenceNode)

  Get the score for recursion order.
- double **Score** (TreeType &queryNode, TreeType &referenceNode)

  Get the score for recursion order.

Private Member Functions

- double **CalculateBound** (TreeType &queryNode) const

  Recalculate the bound for a given query node.
- void **InsertNeighbor** (const size_t queryIndex, const size_t pos, const size_t neighbor, const double distance)

  Insert a point into the neighbors and distances matrices; this is a helper function.

Private Attributes

- arma::mat & **distances**

  The matrix the resultant neighbor distances should be stored in.
- double **lastBaseCase**

  The last base case result.
- size_t **lastQueryIndex**

  *The last query point **BaseCase()** (p. 325) was called with.*
- size_t **lastReferenceIndex**

  *The last reference point **BaseCase()** (p. 325) was called with.*
- MetricType & **metric**

  The instantiated metric.

- arma::Mat< size_t > & **neighbors**

  The matrix the resultant neighbor indices should be stored in.
- const arma::mat & **querySet**

  The query set.
- const arma::mat & **referenceSet**

  The reference set.

21.62.1 Detailed Description

template<typename SortPolicy, typename MetricType, typename TreeType>class mlpack::neighbor::NeighborSearchRules< SortPolicy, MetricType, TreeType >

Definition at line 30 of file neighbor_search_rules.hpp.

21.62.2 Constructor & Destructor Documentation

21.62.2.1 template<typename SortPolicy , typename MetricType , typename TreeType > **mlpack::neighbor::NeighborSearchRules**< SortPolicy, MetricType, TreeType >::**NeighborSearchRules** (const arma::mat & *referenceSet,* const arma::mat & *querySet,* arma::Mat< size_t > & *neighbors,* arma::mat & *distances,* MetricType & *metric*)

21.62.3 Member Function Documentation

21.62.3.1 template<typename SortPolicy , typename MetricType , typename TreeType > double **mlpack::neighbor::NeighborSearchRules**< SortPolicy, MetricType, TreeType >::BaseCase (const size_t *queryIndex,* const size_t *referenceIndex*)

21.62.3.2 template<typename SortPolicy , typename MetricType , typename TreeType > double **mlpack::neighbor::NeighborSearchRules**< SortPolicy, MetricType, TreeType >::CalculateBound (TreeType & *queryNode*) const [private]

Recalculate the bound for a given query node.

21.62.3.3 template<typename SortPolicy , typename MetricType , typename TreeType > void **mlpack::neighbor::NeighborSearchRules**< SortPolicy, MetricType, TreeType >::InsertNeighbor (const size_t *queryIndex,* const size_t *pos,* const size_t *neighbor,* const double *distance*) [private]

Insert a point into the neighbors and distances matrices; this is a helper function.

Parameters

queryIndex	Index of point whose neighbors we are inserting into.
pos	Position in list to insert into.
neighbor	Index of reference point which is being inserted.
distance	Distance from query point to reference point.

21.62.3.4 template<typename SortPolicy , typename MetricType , typename TreeType > double **mlpack::neighbor::NeighborSearchRules**< SortPolicy, MetricType, TreeType >::Rescore (const size_t *queryIndex,* TreeType & *referenceNode,* const double *oldScore*) const

Re-evaluate the score for recursion order.

A low score indicates priority for recursion, while DBL_MAX indicates that the node should not be recursed into at all (it should be pruned). This is used when the score has already been calculated, but another recursion may have modified the bounds for pruning. So the old score is checked against the new pruning bound.

Parameters

queryIndex	Index of query point.
referenceNode	Candidate node to be recursed into.
oldScore	Old score produced by **Score()** (p. 326) (or **Rescore()** (p. 325)).

21.62.3.5 template<typename SortPolicy , typename MetricType , typename TreeType > double mlpack::neighbor::NeighborSearchRules< SortPolicy, MetricType, TreeType >::Rescore (TreeType & *queryNode,* TreeType & *referenceNode,* const double *oldScore*) const

Re-evaluate the score for recursion order.

A low score indicates priority for recursion, while DBL_MAX indicates that the node should not be recursed into at all (it should be pruned). This is used when the score has already been calculated, but another recursion may have modified the bounds for pruning. So the old score is checked against the new pruning bound.

Parameters

queryNode	Candidate query node to recurse into.
referenceNode	Candidate reference node to recurse into.
oldScore	Old score produced by Socre() (or **Rescore()** (p. 325)).

21.62.3.6 template<typename SortPolicy , typename MetricType , typename TreeType > double mlpack::neighbor::NeighborSearchRules< SortPolicy, MetricType, TreeType >::Score (const size_t *queryIndex,* TreeType & *referenceNode*)

Get the score for recursion order.

A low score indicates priority for recursion, while DBL_MAX indicates that the node should not be recursed into at all (it should be pruned).

Parameters

queryIndex	Index of query point.
referenceNode	Candidate node to be recursed into.

21.62.3.7 template<typename SortPolicy , typename MetricType , typename TreeType > double mlpack::neighbor::NeighborSearchRules< SortPolicy, MetricType, TreeType >::Score (TreeType & *queryNode,* TreeType & *referenceNode*)

Get the score for recursion order.

A low score indicates priority for recursionm while DBL_MAX indicates that the node should not be recursed into at all (it should be pruned).

Parameters

queryNode	Candidate query node to recurse into.
referenceNode	Candidate reference node to recurse into.

21.62.4 Member Data Documentation

21.62.4.1 template<typename SortPolicy , typename MetricType , typename TreeType > arma::mat& mlpack::neighbor::NeighborSearchRules< SortPolicy, MetricType, TreeType >::distances [private]

The matrix the resultant neighbor distances should be stored in.

Definition at line 102 of file neighbor_search_rules.hpp.

21.62.4.2 template<typename SortPolicy , typename MetricType , typename TreeType > double mlpack::neighbor::NeighborSearchRules< SortPolicy, MetricType, TreeType >::lastBaseCase [private]

The last base case result.

Definition at line 112 of file neighbor_search_rules.hpp.

21.62.4.3 template<typename SortPolicy , typename MetricType , typename TreeType > size_t mlpack::neighbor::NeighborSearchRules< SortPolicy, MetricType, TreeType >::lastQueryIndex [private]

The last query point **BaseCase()** (p. 325) was called with.

Definition at line 108 of file neighbor_search_rules.hpp.

21.62.4.4 template<typename SortPolicy , typename MetricType , typename TreeType > size_t mlpack::neighbor::NeighborSearchRules< SortPolicy, MetricType, TreeType >::lastReferenceIndex [private]

The last reference point **BaseCase()** (p. 325) was called with.

Definition at line 110 of file neighbor_search_rules.hpp.

21.62.4.5 template<typename SortPolicy , typename MetricType , typename TreeType > MetricType& mlpack::neighbor::NeighborSearchRules< SortPolicy, MetricType, TreeType >::metric [private]

The instantiated metric.

Definition at line 105 of file neighbor_search_rules.hpp.

21.62.4.6 template<typename SortPolicy , typename MetricType , typename TreeType > arma::Mat<size_t>& mlpack::neighbor::NeighborSearchRules< SortPolicy, MetricType, TreeType >::neighbors [private]

The matrix the resultant neighbor indices should be stored in.

Definition at line 99 of file neighbor_search_rules.hpp.

21.62.4.7 template<typename SortPolicy , typename MetricType , typename TreeType > const arma::mat& mlpack::neighbor::NeighborSearchRules< SortPolicy, MetricType, TreeType >::querySet [private]

The query set.

Definition at line 96 of file neighbor_search_rules.hpp.

21.62.4.8 template<typename SortPolicy , typename MetricType , typename TreeType > const arma::mat& mlpack::neighbor::NeighborSearchRules< SortPolicy, MetricType, TreeType >::referenceSet [private]

The reference set.

Definition at line 93 of file neighbor_search_rules.hpp.

The documentation for this class was generated from the following file:

- src/mlpack/methods/neighbor_search/**neighbor_search_rules.hpp**

21.63 mlpack::neighbor::NeighborSearchStat< SortPolicy > Class Template Reference

Extra data for each node in the tree.

Public Member Functions

- **NeighborSearchStat** ()

  Initialize the statistic with the worst possible distance according to our sorting policy.
- template<typename TreeType >
  NeighborSearchStat (TreeType &)

  Initialization for a fully initialized node.
- double **Bound** () const

  Get the overall bound (the better of the two bounds).
- double & **Bound** ()

  Modify the overall bound (it should be the better of the two bounds).
- double **FirstBound** () const

  Get the first bound.
- double & **FirstBound** ()

  Modify the first bound.
- double **LastDistance** () const

  Get the last distance calculation.
- double & **LastDistance** ()

  Modify the last distance calculation.
- void * **LastDistanceNode** () const

  Get the last distance evaluation node.
- void *& **LastDistanceNode** ()

  Modify the last distance evaluation node.
- double **SecondBound** () const

  Get the second bound.
- double & **SecondBound** ()

  Modify the second bound.

Private Attributes

- double **bound**

  The better of the two bounds.
- double **firstBound**

  The first bound on the node's neighbor distances (B_1).
- double **lastDistance**

  The last distance evaluation.
- void * **lastDistanceNode**

  The last distance evaluation node.
- double **secondBound**

  The second bound on the node's neighbor distances (B_2).

21.63.1 Detailed Description

template<typename SortPolicy>class mlpack::neighbor::NeighborSearchStat< SortPolicy >

Extra data for each node in the tree.

For neighbor searches, each node only needs to store a bound on neighbor distances.

Definition at line 36 of file neighbor_search_stat.hpp.

21.63.2 Constructor & Destructor Documentation

21.63.2.1 template<typename SortPolicy > mlpack::neighbor::NeighborSearchStat< SortPolicy >::NeighborSearchStat ( ) [inline]

Initialize the statistic with the worst possible distance according to our sorting policy.

Definition at line 60 of file neighbor_search_stat.hpp.

21.63.2.2 template<typename SortPolicy > template<typename TreeType > mlpack::neighbor::NeighborSearchStat< SortPolicy >::NeighborSearchStat (TreeType &) [inline]

Initialization for a fully initialized node.

In this case, we don't need to worry about the node.

Definition at line 72 of file neighbor_search_stat.hpp.

21.63.3 Member Function Documentation

21.63.3.1 template<typename SortPolicy > double mlpack::neighbor::NeighborSearchStat< SortPolicy >::Bound ( ) const [inline]

Get the overall bound (the better of the two bounds).

Definition at line 88 of file neighbor_search_stat.hpp.

References mlpack::neighbor::NeighborSearchStat< SortPolicy >::bound.

21.63.3.2 template<typename SortPolicy> double& mlpack::neighbor::NeighborSearchStat< SortPolicy >::Bound ( ) [inline]

Modify the overall bound (it should be the better of the two bounds).

Definition at line 90 of file neighbor_search_stat.hpp.

References mlpack::neighbor::NeighborSearchStat< SortPolicy >::bound.

21.63.3.3 template<typename SortPolicy> double mlpack::neighbor::NeighborSearchStat< SortPolicy >::FirstBound ( ) const [inline]

Get the first bound.

Definition at line 80 of file neighbor_search_stat.hpp.

References mlpack::neighbor::NeighborSearchStat< SortPolicy >::firstBound.

21.63.3.4 template<typename SortPolicy> double& mlpack::neighbor::NeighborSearchStat< SortPolicy >::FirstBound ( ) [inline]

Modify the first bound.

Definition at line 82 of file neighbor_search_stat.hpp.

References mlpack::neighbor::NeighborSearchStat< SortPolicy >::firstBound.

21.63.3.5 template<typename SortPolicy> double mlpack::neighbor::NeighborSearchStat< SortPolicy >::LastDistance ( ) const [inline]

Get the last distance calculation.

Definition at line 96 of file neighbor_search_stat.hpp.

References mlpack::neighbor::NeighborSearchStat< SortPolicy >::lastDistance.

21.63.3.6 template<typename SortPolicy> double& mlpack::neighbor::NeighborSearchStat< SortPolicy >::LastDistance ( ) [inline]

Modify the last distance calculation.

Definition at line 98 of file neighbor_search_stat.hpp.

References mlpack::neighbor::NeighborSearchStat< SortPolicy >::lastDistance.

21.63.3.7 template<typename SortPolicy> void* mlpack::neighbor::NeighborSearchStat< SortPolicy >::LastDistanceNode ( ) const [inline]

Get the last distance evaluation node.

Definition at line 92 of file neighbor_search_stat.hpp.

References mlpack::neighbor::NeighborSearchStat< SortPolicy >::lastDistanceNode.

21.63.3.8 template<typename SortPolicy > void*& mlpack::neighbor::NeighborSearchStat< SortPolicy >::LastDistanceNode ( ) [inline]

Modify the last distance evaluation node.

Definition at line 94 of file neighbor_search_stat.hpp.

References mlpack::neighbor::NeighborSearchStat< SortPolicy >::lastDistanceNode.

21.63.3.9 template<typename SortPolicy > double mlpack::neighbor::NeighborSearchStat< SortPolicy >::SecondBound ( ) const [inline]

Get the second bound.

Definition at line 84 of file neighbor_search_stat.hpp.

References mlpack::neighbor::NeighborSearchStat< SortPolicy >::secondBound.

21.63.3.10 template<typename SortPolicy > double& mlpack::neighbor::NeighborSearchStat< SortPolicy >::SecondBound ( ) [inline]

Modify the second bound.

Definition at line 86 of file neighbor_search_stat.hpp.

References mlpack::neighbor::NeighborSearchStat< SortPolicy >::secondBound.

21.63.4 Member Data Documentation

21.63.4.1 template<typename SortPolicy > double mlpack::neighbor::NeighborSearchStat< SortPolicy >::bound [private]

The better of the two bounds.

Definition at line 48 of file neighbor_search_stat.hpp.

Referenced by mlpack::neighbor::NeighborSearchStat< SortPolicy >::Bound().

21.63.4.2 template<typename SortPolicy > double mlpack::neighbor::NeighborSearchStat< SortPolicy >::firstBound [private]

The first bound on the node's neighbor distances (B_1).

This represents the worst candidate distance of any descendants of this node.

Definition at line 41 of file neighbor_search_stat.hpp.

Referenced by mlpack::neighbor::NeighborSearchStat< SortPolicy >::FirstBound().

21.63.4.3 template<typename SortPolicy > double mlpack::neighbor::NeighborSearchStat< SortPolicy >::lastDistance [private]

The last distance evaluation.

Definition at line 53 of file neighbor_search_stat.hpp.

Referenced by mlpack::neighbor::NeighborSearchStat< SortPolicy >::LastDistance().

21.63.4.4 template<typename SortPolicy > void* mlpack::neighbor::NeighborSearchStat< SortPolicy >::lastDistanceNode [private]

The last distance evaluation node.

Definition at line 51 of file neighbor_search_stat.hpp.

Referenced by mlpack::neighbor::NeighborSearchStat< SortPolicy >::LastDistanceNode().

21.63.4.5 template<typename SortPolicy > double mlpack::neighbor::NeighborSearchStat< SortPolicy >::secondBound [private]

The second bound on the node's neighbor distances (B_2).

This represents a bound on the worst distance of any descendants of this node assembled using the best descendant candidate distance modified by the furthest descendant distance.

Definition at line 46 of file neighbor_search_stat.hpp.

Referenced by mlpack::neighbor::NeighborSearchStat< SortPolicy >::SecondBound().

The documentation for this class was generated from the following file:

- src/mlpack/methods/neighbor_search/**neighbor_search_stat.hpp**

21.64 mlpack::neighbor::RAQueryStat< SortPolicy > Class Template Reference

Extra data for each node in the tree.

Public Member Functions

- **RAQueryStat** ()

  Initialize the statistic with the worst possible distance according to our sorting policy.
- template<typename TreeType >
  RAQueryStat (const TreeType &)

  Initialization for a node.
- double **Bound** () const

  Get the bound.
- double & **Bound** ()

  Modify the bound.
- size_t **NumSamplesMade** () const

  Get the number of samples made.
- size_t & **NumSamplesMade** ()

  Modify the number of samples made.

Private Attributes

- double **bound**

  The bound on the node's neighbor distances.
- size_t **numSamplesMade**

  The minimum number of samples made by any query in this node.

21.64.1 Detailed Description

template<typename SortPolicy>class mlpack::neighbor::RAQueryStat< SortPolicy >

Extra data for each node in the tree.

For neighbor searches, each node only needs to store a bound on neighbor distances.

Every query is required to make a minimum number of samples to guarantee the desired approximation error. The 'numSamplesMade' keeps track of the minimum number of samples made by all queries in the node in question.

Definition at line 57 of file ra_search.hpp.

21.64.2 Constructor & Destructor Documentation

21.64.2.1 template<typename SortPolicy > mlpack::neighbor::RAQueryStat< SortPolicy >::RAQueryStat ( ) [inline]

Initialize the statistic with the worst possible distance according to our sorting policy.

Definition at line 71 of file ra_search.hpp.

21.64.2.2 template<typename SortPolicy > template<typename TreeType > mlpack::neighbor::RAQueryStat< SortPolicy >::RAQueryStat (const TreeType &) [inline]

Initialization for a node.

Definition at line 77 of file ra_search.hpp.

21.64.3 Member Function Documentation

21.64.3.1 template<typename SortPolicy > double mlpack::neighbor::RAQueryStat< SortPolicy >::Bound ( ) const [inline]

Get the bound.

Definition at line 83 of file ra_search.hpp.

References mlpack::neighbor::RAQueryStat< SortPolicy >::bound.

21.64.3.2 template<typename SortPolicy > double& mlpack::neighbor::RAQueryStat< SortPolicy >::Bound ( ) [inline]

Modify the bound.

Definition at line 85 of file ra_search.hpp.

References mlpack::neighbor::RAQueryStat< SortPolicy >::bound.

21.64.3.3 template<typename SortPolicy > size_t mlpack::neighbor::RAQueryStat< SortPolicy >::NumSamplesMade ( ) const [inline]

Get the number of samples made.

Definition at line 88 of file ra_search.hpp.

References mlpack::neighbor::RAQueryStat< SortPolicy >::numSamplesMade.

21.64.3.4 template<typename SortPolicy > size_t& mlpack::neighbor::RAQueryStat< SortPolicy >::NumSamplesMade ( ) [inline]

Modify the number of samples made.

Definition at line 90 of file ra_search.hpp.

References mlpack::neighbor::RAQueryStat< SortPolicy >::numSamplesMade.

21.64.4 Member Data Documentation

21.64.4.1 template<typename SortPolicy > double mlpack::neighbor::RAQueryStat< SortPolicy >::bound [private]

The bound on the node's neighbor distances.

Definition at line 61 of file ra_search.hpp.

Referenced by mlpack::neighbor::RAQueryStat< SortPolicy >::Bound().

21.64.4.2 template<typename SortPolicy > size_t mlpack::neighbor::RAQueryStat< SortPolicy >::numSamplesMade [private]

The minimum number of samples made by any query in this node.

Definition at line 64 of file ra_search.hpp.

Referenced by mlpack::neighbor::RAQueryStat< SortPolicy >::NumSamplesMade().

The documentation for this class was generated from the following file:

- src/mlpack/methods/rann/**ra_search.hpp**

21.65 mlpack::neighbor::RASearch< SortPolicy, MetricType, TreeType > Class Template Reference

The **RASearch** (p. 334) class: This class provides a generic manner to perform rank-approximate search via random-sampling.

Public Member Functions

- **RASearch** (const typename TreeType::Mat &**referenceSet**, const typename TreeType::Mat &**querySet**, const bool **naive**=false, const bool **singleMode**=false, const size_t leafSize=20, const MetricType **metric**=MetricType())

  *Initialize the **RASearch** (p. 334) object, passing both a query and reference dataset.*
- **RASearch** (const typename TreeType::Mat &**referenceSet**, const bool **naive**=false, const bool **singleMode**=false, const size_t leafSize=20, const MetricType **metric**=MetricType())

  *Initialize the **RASearch** (p. 334) object, passing only one dataset, which is used as both the query and the reference dataset.*
- **RASearch** (TreeType *__referenceTree__, TreeType *__queryTree__, const typename TreeType::Mat &**referenceSet**, const typename TreeType::Mat &**querySet**, const bool **singleMode**=false, const MetricType **metric**=MetricType())

*Initialize the **RASearch** (p. 334) object with the given datasets and pre-constructed trees.*

- **RASearch** (TreeType *__referenceTree__, const typename TreeType::Mat &__referenceSet__, const bool __singleMode__=false, const MetricType __metric__=MetricType())

  *Initialize the **RASearch** (p. 334) object with the given reference dataset and pre-constructed tree.*
- ~**RASearch** ()

  *Delete the **RASearch** (p. 334) object.*
- void **ResetQueryTree** ()

  *This function recursively resets the **RAQueryStat** (p. 332) of the queryTree to set 'bound' to WorstDistance and the 'numSamplesMade' to 0.*
- void **Search** (const size_t k, arma::Mat< size_t > &resultingNeighbors, arma::mat &distances, const double tau=5, const double __alpha__=0.95, const bool sampleAtLeaves=false, const bool firstLeafExact=false, const size_t singleSampleLimit=20)

  Compute the rank approximate nearest neighbors and store the output in the given matrices.

Private Member Functions

- void **ResetRAQueryStat** (TreeType *treeNode)

Private Attributes

- MetricType **metric**

  Instantiation of kernel.
- bool **naive**

  Indicates if naive random sampling on the set is being used.
- size_t **numberOfPrunes**

  Total number of pruned nodes during the neighbor search.
- std::vector< size_t > **oldFromNewQueries**

  Permutations of query points during tree building.
- std::vector< size_t > **oldFromNewReferences**

  Permutations of reference points during tree building.
- bool **ownQueryTree**

  Indicates if we should free the query tree at deletion time.
- bool **ownReferenceTree**

  Indicates if we should free the reference tree at deletion time.
- arma::mat **queryCopy**

  Copy of query dataset (if we need it, because tree building modifies it).
- const arma::mat & **querySet**

  Query dataset (may not be given).
- TreeType * **queryTree**

  Pointer to the root of the query tree (might not exist).
- arma::mat **referenceCopy**

  Copy of reference dataset (if we need it, because tree building modifies it).
- const arma::mat & **referenceSet**

  Reference dataset.
- TreeType * **referenceTree**

  Pointer to the root of the reference tree.
- bool **singleMode**

  Indicates if single-tree search is being used (opposed to dual-tree).

21.65.1 Detailed Description

template<typename SortPolicy = NearestNeighborSort, typename MetricType = mlpack::metric::SquaredEuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2, false>, RAQueryStat<SortPolicy> >>class mlpack::neighbor::RASearch< SortPolicy, MetricType, TreeType >

The **RASearch** (p. 334) class: This class provides a generic manner to perform rank-approximate search via random-sampling.

If the 'naive' option is chosen, this rank-approximate search will be done by randomly sampled from the whole set. If the 'naive' option is not chosen, the sampling is done in a stratified manner in the tree as mentioned in the algorithms in Figure 2 of the following paper:

{ram2009rank, title={{Rank-Approximate Nearest Neighbor Search: Retaining Meaning and Speed in High Dimensions}}, author={{Ram, P. and Lee, D. and Ouyang, H. and Gray, A. G.}}, booktitle={{Advances of Neural Information Processing Systems}}, year={2009} }

Template Parameters

SortPolicy	The sort policy for distances; see **NearestNeighborSort** (p. 314).
MetricType	The metric to use for computation.
TreeType	The tree type to use.

Definition at line 117 of file ra_search.hpp.

21.65.2 Constructor & Destructor Documentation

21.65.2.1 template<typename SortPolicy = NearestNeighborSort, typename MetricType = mlpack::metric::SquaredEuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2, false>, RAQueryStat<SortPolicy> >> mlpack::neighbor::RASearch< SortPolicy, MetricType, TreeType >::RASearch (const typename TreeType::Mat & *referenceSet,* const typename TreeType::Mat & *querySet,* const bool *naive* = `false`, const bool *singleMode* = `false`, const size_t *leafSize* = `20`, const MetricType *metric* = `MetricType()`)

Initialize the **RASearch** (p. 334) object, passing both a query and reference dataset.

Optionally, perform the computation in naive mode or single-tree mode, and set the leaf size used for tree-building. An initialized distance metric can be given, for cases where the metric has internal data (i.e. the distance::MahalanobisDistance class).

This method will copy the matrices to internal copies, which are rearranged during tree-building. You can avoid this extra copy by pre-constructing the trees and passing them using a diferent constructor.

Parameters

referenceSet	Set of reference points.
querySet	Set of query points.
naive	If true, the rank-approximate search will be performed by directly sampling the whole set instead of using the stratified sampling on the tree.
singleMode	If true, single-tree search will be used (as opposed to dual-tree search).
leafSize	Leaf size for tree construction (ignored if tree is given).
metric	An optional instance of the MetricType class.

21.65.2.2 template<typename SortPolicy = NearestNeighborSort, typename MetricType = mlpack::metric::SquaredEuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2, false>, RAQueryStat<SortPolicy> >> mlpack::neighbor::RASearch< SortPolicy, MetricType, TreeType >::RASearch (const typename TreeType::Mat & *referenceSet,* const bool *naive* = `false`, const bool *singleMode* = `false`, const size_t *leafSize* = `20`, const MetricType *metric* = `MetricType()`)

Initialize the **RASearch** (p. 334) object, passing only one dataset, which is used as both the query and the reference dataset.

Optionally, perform the computation in naive mode or single-tree mode, and set the leaf size used for tree-building. An initialized distance metric can be given, for cases where the metric has internal data (i.e. the distance::MahalanobisDistance class).

If naive mode is being used and a pre-built tree is given, it may not work: naive mode operates by building a one-node tree (the root node holds all the points). If that condition is not satisfied with the pre-built tree, then naive mode will not work.

Parameters

referenceSet	Set of reference points.
naive	If true, the rank-approximate search will be performed by directly sampling the whole set instead of using the stratified sampling on the tree.
singleMode	If true, single-tree search will be used (as opposed to dual-tree search).
leafSize	Leaf size for tree construction (ignored if tree is given).
metric	An optional instance of the MetricType class.

21.65.2.3 template<typename SortPolicy = NearestNeighborSort, typename MetricType = mlpack::metric::SquaredEuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2, false>, RAQueryStat<SortPolicy> >> mlpack::neighbor::RASearch< SortPolicy, MetricType, TreeType >::RASearch (TreeType * *referenceTree,* TreeType * *queryTree,* const typename TreeType::Mat & *referenceSet,* const typename TreeType::Mat & *querySet,* const bool *singleMode* = `false`, const MetricType *metric* = `MetricType()`)

Initialize the **RASearch** (p. 334) object with the given datasets and pre-constructed trees.

It is assumed that the points in referenceSet and querySet correspond to the points in referenceTree and queryTree, respectively. Optionally, choose to use single-tree mode. Naive mode is not available as an option for this constructor; instead, to run naive computation, construct a tree with all of the points in one leaf (i.e. leafSize = number of points). Additionally, an instantiated distance metric can be given, for cases where the distance metric holds data.

There is no copying of the data matrices in this constructor (because tree-building is not necessary), so this is the constructor to use when copies absolutely must be avoided.

Note

Because tree-building (at least with BinarySpaceTree) modifies the ordering of a matrix, be sure you pass the modified matrix to this object! In addition, mapping the points of the matrix back to their original indices is not done when this constructor is used.

Parameters

referenceTree	Pre-built tree for reference points.
queryTree	Pre-built tree for query points.

referenceSet	Set of reference points corresponding to referenceTree.
querySet	Set of query points corresponding to queryTree.
singleMode	Whether single-tree computation should be used (as opposed to dual-tree computation).
metric	Instantiated distance metric.

21.65.2.4 template<typename SortPolicy = NearestNeighborSort, typename MetricType = mlpack::metric::SquaredEuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2, false>, RAQueryStat<SortPolicy> >> mlpack::neighbor::RASearch< SortPolicy, MetricType, TreeType >::RASearch (TreeType * *referenceTree,* const typename TreeType::Mat & *referenceSet,* const bool *singleMode* = `false`, const MetricType *metric* = `MetricType()`)

Initialize the **RASearch** (p. 334) object with the given reference dataset and pre-constructed tree.

It is assumed that the points in referenceSet correspond to the points in referenceTree. Optionally, choose to use single-tree mode. Naive mode is not available as an option for this constructor; instead, to run naive computation, construct a tree with all the points in one leaf (i.e. leafSize = number of points). Additionally, an instantiated distance metric can be given, for the case where the distance metric holds data.

There is no copying of the data matrices in this constructor (because tree-building is not necessary), so this is the constructor to use when copies absolutely must be avoided.

Note

Because tree-building (at least with BinarySpaceTree) modifies the ordering of a matrix, be sure you pass the modified matrix to this object! In addition, mapping the points of the matrix back to their original indices is not done when this constructor is used.

Parameters

referenceTree	Pre-built tree for reference points.
referenceSet	Set of reference points corresponding to referenceTree.
singleMode	Whether single-tree computation should be used (as opposed to dual-tree computation).
metric	Instantiated distance metric.

21.65.2.5 template<typename SortPolicy = NearestNeighborSort, typename MetricType = mlpack::metric::SquaredEuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2, false>, RAQueryStat<SortPolicy> >> mlpack::neighbor::RASearch< SortPolicy, MetricType, TreeType >::~RASearch ( )

Delete the **RASearch** (p. 334) object.

The tree is the only member we are responsible for deleting. The others will take care of themselves.

21.65.3 Member Function Documentation

21.65.3.1 template<typename SortPolicy = NearestNeighborSort, typename MetricType = mlpack::metric::SquaredEuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2, false>, RAQueryStat<SortPolicy> >> void mlpack::neighbor::RASearch< SortPolicy, MetricType, TreeType >::ResetQueryTree ( )

This function recursively resets the **RAQueryStat** (p. 332) of the queryTree to set 'bound' to WorstDistance and the 'numSamplesMade' to 0.

This allows a user to perform multiple searches on the same pair of trees, possibly with different levels of approximation without requiring to build a new pair of trees for every new (approximate) search.

21.65.3.2 template<typename SortPolicy = NearestNeighborSort, typename MetricType = mlpack::metric::SquaredEuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2, false>, RAQueryStat<SortPolicy> >> void mlpack::neighbor::RASearch< SortPolicy, MetricType, TreeType >::ResetRAQueryStat (TreeType * *treeNode*) [private]

Parameters

treeNode	The node of the tree whose **RAQueryStat** (p. 332) is reset and whose children are to be explored recursively.

21.65.3.3 template<typename SortPolicy = NearestNeighborSort, typename MetricType = mlpack::metric::SquaredEuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2, false>, RAQueryStat<SortPolicy> >> void mlpack::neighbor::RASearch< SortPolicy, MetricType, TreeType >::Search (const size_t *k*, arma::Mat< size_t > & *resultingNeighbors*, arma::mat & *distances*, const double *tau* = 5, const double *alpha* = 0.95, const bool *sampleAtLeaves* = false, const bool *firstLeafExact* = false, const size_t *singleSampleLimit* = 20)

Compute the rank approximate nearest neighbors and store the output in the given matrices.

The matrices will be set to the size of n columns by k rows, where n is the number of points in the query dataset and k is the number of neighbors being searched for.

Note that tau, the rank-approximation parameter, specifies that we are looking for k neighbors with probability alpha of being in the top tau percent of nearest neighbors. So, as an example, if our dataset has 1000 points, and we want 5 nearest neighbors with 95% probability of being in the top 5% of nearest neighbors (or, the top 50 nearest neighbors), we set k = 5, tau = 5, and alpha = 0.95.

The method will fail (and issue a failure message) if the value of tau is too low: tau must be set such that the number of points in the corresponding percentile of the data is greater than k. Thus, if we choose tau = 0.1 with a dataset of 1000 points and k = 5, then we are attempting to choose 5 nearest neighbors out of the closest 1 point – this is invalid.

Parameters

k	Number of neighbors to search for.
resulting-Neighbors	Matrix storing lists of neighbors for each query point.
distances	Matrix storing distances of neighbors for each query point.
tau	The rank-approximation in percentile of the data. The default value is 5%.
alpha	The desired success probability. The default value is 0.95.
sampleAtLeaves	Sample at leaves for faster but less accurate computation. This defaults to 'false'.
firstLeafExact	Traverse to the first leaf without approximation. This can ensure that the query definitely finds its (near) duplicate if there exists one. This defaults to 'false' for now.
singleSample-Limit	The limit on the largest node that can be approximated by sampling. This defaults to 20.

21.65.4 Member Data Documentation

21.65.4.1 template<typename SortPolicy = NearestNeighborSort, typename MetricType = mlpack::metric::SquaredEuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2, false>, RAQueryStat<SortPolicy> >> MetricType mlpack::neighbor::RASearch< SortPolicy, MetricType, TreeType >::metric [private]

Instantiation of kernel.

Definition at line 332 of file ra_search.hpp.

21.65.4.2 template<typename SortPolicy = NearestNeighborSort, typename MetricType = mlpack::metric::SquaredEuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2, false>, RAQueryStat<SortPolicy> >> bool mlpack::neighbor::RASearch< SortPolicy, MetricType, TreeType >::naive [private]

Indicates if naive random sampling on the set is being used.

Definition at line 327 of file ra_search.hpp.

21.65.4.3 template<typename SortPolicy = NearestNeighborSort, typename MetricType = mlpack::metric::SquaredEuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2, false>, RAQueryStat<SortPolicy> >> size_t mlpack::neighbor::RASearch< SortPolicy, MetricType, TreeType >::numberOfPrunes [private]

Total number of pruned nodes during the neighbor search.

Definition at line 340 of file ra_search.hpp.

21.65.4.4 template<typename SortPolicy = NearestNeighborSort, typename MetricType = mlpack::metric::SquaredEuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2, false>, RAQueryStat<SortPolicy> >> std::vector<size_t> mlpack::neighbor::RASearch< SortPolicy, MetricType, TreeType >::oldFromNewQueries [private]

Permutations of query points during tree building.

Definition at line 337 of file ra_search.hpp.

21.65.4.5 template<typename SortPolicy = NearestNeighborSort, typename MetricType = mlpack::metric::SquaredEuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2, false>, RAQueryStat<SortPolicy> >> std::vector<size_t> mlpack::neighbor::RASearch< SortPolicy, MetricType, TreeType >::oldFromNewReferences [private]

Permutations of reference points during tree building.

Definition at line 335 of file ra_search.hpp.

21.65.4.6 template<typename SortPolicy = NearestNeighborSort, typename MetricType = mlpack::metric::SquaredEuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2, false>, RAQueryStat<SortPolicy> >> bool mlpack::neighbor::RASearch< SortPolicy, MetricType, TreeType >::ownQueryTree [private]

Indicates if we should free the query tree at deletion time.

Definition at line 324 of file ra_search.hpp.

21.65.4.7 template<typename SortPolicy = NearestNeighborSort, typename MetricType = mlpack::metric::SquaredEuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2, false>, RAQueryStat<SortPolicy> >> bool mlpack::neighbor::RASearch< SortPolicy, MetricType, TreeType >::ownReferenceTree [private]

Indicates if we should free the reference tree at deletion time.

Definition at line 322 of file ra_search.hpp.

21.65.4.8 template<typename SortPolicy = NearestNeighborSort, typename MetricType = mlpack::metric::SquaredEuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2, false>, RAQueryStat<SortPolicy> >> arma::mat mlpack::neighbor::RASearch< SortPolicy, MetricType, TreeType >::queryCopy [private]

Copy of query dataset (if we need it, because tree building modifies it).

Definition at line 309 of file ra_search.hpp.

21.65.4.9 template<typename SortPolicy = NearestNeighborSort, typename MetricType = mlpack::metric::SquaredEuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2, false>, RAQueryStat<SortPolicy> >> const arma::mat& mlpack::neighbor::RASearch< SortPolicy, MetricType, TreeType >::querySet [private]

Query dataset (may not be given).

Definition at line 314 of file ra_search.hpp.

21.65.4.10 template<typename SortPolicy = NearestNeighborSort, typename MetricType = mlpack::metric::SquaredEuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2, false>, RAQueryStat<SortPolicy> >> TreeType* mlpack::neighbor::RASearch< SortPolicy, MetricType, TreeType >::queryTree [private]

Pointer to the root of the query tree (might not exist).

Definition at line 319 of file ra_search.hpp.

21.65.4.11 template<typename SortPolicy = NearestNeighborSort, typename MetricType = mlpack::metric::SquaredEuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2, false>, RAQueryStat<SortPolicy> >> arma::mat mlpack::neighbor::RASearch< SortPolicy, MetricType, TreeType >::referenceCopy [private]

Copy of reference dataset (if we need it, because tree building modifies it).

Definition at line 307 of file ra_search.hpp.

21.65.4.12 template<typename SortPolicy = NearestNeighborSort, typename MetricType = mlpack::metric::SquaredEuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2, false>, RAQueryStat<SortPolicy> >> const arma::mat& mlpack::neighbor::RASearch< SortPolicy, MetricType, TreeType >::referenceSet [private]

Reference dataset.

Definition at line 312 of file ra_search.hpp.

21.65.4.13 template<typename SortPolicy = NearestNeighborSort, typename MetricType = mlpack::metric::SquaredEuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2, false>, RAQueryStat<SortPolicy> >> TreeType* mlpack::neighbor::RASearch< SortPolicy, MetricType, TreeType >::referenceTree [private]

Pointer to the root of the reference tree.

Definition at line 317 of file ra_search.hpp.

21.65.4.14 template<typename SortPolicy = NearestNeighborSort, typename MetricType = mlpack::metric::SquaredEuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2, false>, RAQueryStat<SortPolicy> >> bool mlpack::neighbor::RASearch< SortPolicy, MetricType, TreeType >::singleMode [private]

Indicates if single-tree search is being used (opposed to dual-tree).

Definition at line 329 of file ra_search.hpp.

The documentation for this class was generated from the following file:

- src/mlpack/methods/rann/**ra_search.hpp**

21.66 mlpack::neighbor::RASearchRules< SortPolicy, MetricType, TreeType > Class Template Reference

Public Member Functions

- **RASearchRules** (const arma::mat &**referenceSet**, const arma::mat &**querySet**, arma::Mat< size_t > &**neighbors**, arma::mat &**distances**, MetricType &**metric**, const double tau=5, const double **alpha**=0.95, const bool naive=false, const bool **sampleAtLeaves**=false, const bool **firstLeafExact**=false, const size_t **singleSampleLimit**=20)
- double **BaseCase** (const size_t queryIndex, const size_t referenceIndex)
- size_t **NumDistComputations** ()
- size_t **NumEffectiveSamples** ()
- double **Prescore** (TreeType &queryNode, TreeType &referenceNode, TreeType &referenceChildNode, const double baseCaseResult) const

  TOFIX: This function is specified for the cover tree (usually) so I need to think about it more algorithmically and keep its implementation mostly empty.
- double **PrescoreQ** (TreeType &queryNode, TreeType &queryChildNode, TreeType &referenceNode, const double baseCaseResult) const
- double **Rescore** (const size_t queryIndex, TreeType &referenceNode, const double oldScore)

  Re-evaluate the score for recursion order.
- double **Rescore** (TreeType &queryNode, TreeType &referenceNode, const double oldScore)

  Re-evaluate the score for recursion order.
- double **Score** (const size_t queryIndex, TreeType &referenceNode)

  Get the score for recursion order.
- double **Score** (const size_t queryIndex, TreeType &referenceNode, const double baseCaseResult)

  Get the score for recursion order.
- double **Score** (TreeType &queryNode, TreeType &referenceNode)

  Get the score for recursion order.
- double **Score** (TreeType &queryNode, TreeType &referenceNode, const double baseCaseResult)

  Get the score for recursion order, passing the base case result (in the situation where it may be needed to calculate the recursion order).

Private Member Functions

- void **InsertNeighbor** (const size_t queryIndex, const size_t pos, const size_t neighbor, const double distance)

  Insert a point into the neighbors and distances matrices; this is a helper function.
- size_t **MinimumSamplesReqd** (const size_t n, const size_t k, const double tau, const double **alpha**) const

Compute the minimum number of samples required to guarantee the given rank-approximation and success probability.

- void **ObtainDistinctSamples** (const size_t numSamples, const size_t rangeUpperBound, arma::uvec &distinctSamples) const

  Pick up desired number of samples (with replacement) from a given range of integers so that only the distinct samples are returned from the range [0 - specified upper bound)
- double **Score** (const size_t queryIndex, TreeType &referenceNode, const double distance, const double bestDistance)

  Perform actual scoring for single-tree case.
- double **Score** (TreeType &queryNode, TreeType &referenceNode, const double distance, const double bestDistance)

  Perform actual scoring for dual-tree case.
- double **SuccessProbability** (const size_t n, const size_t k, const size_t m, const size_t t) const

  Compute the success probability of obtaining 'k'-neighbors from a set of size 'n' within the top 't' neighbors if 'm' samples are made.

Private Attributes

- arma::mat & **distances**

  The matrix the resultant neighbor distances should be stored in.
- bool **firstLeafExact**

  Whether to do exact computation on the first leaf before any sampling.
- MetricType & **metric**

  The instantiated metric.
- arma::Mat< size_t > & **neighbors**

  The matrix the resultant neighbor indices should be stored in.
- size_t **numDistComputations**
- arma::Col< size_t > **numSamplesMade**

  The number of samples made for every query.
- size_t **numSamplesReqd**

  The minimum number of samples required per query.
- const arma::mat & **querySet**

  The query set.
- const arma::mat & **referenceSet**

  The reference set.
- bool **sampleAtLeaves**

  Whether to sample at leaves or just use all of it.
- double **samplingRatio**

  The sampling ratio.
- size_t **singleSampleLimit**

  The limit on the largest node that can be approximated by sampling.

21.66.1 Detailed Description

template<typename SortPolicy, typename MetricType, typename TreeType>class mlpack::neighbor::RASearchRules< SortPolicy, MetricType, TreeType >

Definition at line 31 of file ra_search_rules.hpp.

21.66.2 Constructor & Destructor Documentation

21.66.2.1 template<typename SortPolicy , typename MetricType , typename TreeType > **mlpack::neighbor::RASearchRules**< SortPolicy, MetricType, TreeType >::**RASearchRules** (const arma::mat & *referenceSet,* const arma::mat & *querySet,* arma::Mat< size_t > & *neighbors,* arma::mat & *distances,* MetricType & *metric,* const double *tau* = `5`, const double *alpha* = `0.95`, const bool *naive* = `false`, const bool *sampleAtLeaves* = `false`, const bool *firstLeafExact* = `false`, const size_t *singleSampleLimit* = `20`)

21.66.3 Member Function Documentation

21.66.3.1 template<typename SortPolicy , typename MetricType , typename TreeType > double **mlpack::neighbor::RASearchRules**< SortPolicy, MetricType, TreeType >::**BaseCase** (const size_t *queryIndex,* const size_t *referenceIndex*)

21.66.3.2 template<typename SortPolicy , typename MetricType , typename TreeType > void **mlpack::neighbor::RASearchRules**< SortPolicy, MetricType, TreeType >::**InsertNeighbor** (const size_t *queryIndex,* const size_t *pos,* const size_t *neighbor,* const double *distance*) `[private]`

Insert a point into the neighbors and distances matrices; this is a helper function.

Parameters

queryIndex	Index of point whose neighbors we are inserting into.
pos	Position in list to insert into.
neighbor	Index of reference point which is being inserted.
distance	Distance from query point to reference point.

21.66.3.3 template<typename SortPolicy , typename MetricType , typename TreeType > size_t **mlpack::neighbor::RASearchRules**< SortPolicy, MetricType, TreeType >::**MinimumSamplesReqd** (const size_t *n,* const size_t *k,* const double *tau,* const double *alpha*) const `[private]`

Compute the minimum number of samples required to guarantee the given rank-approximation and success probability.

Parameters

n	Size of the set to be sampled from.
k	The number of neighbors required within the rank-approximation.
tau	The rank-approximation in percentile of the data.
alpha	The success probability desired.

21.66.3.4 template<typename SortPolicy , typename MetricType , typename TreeType > size_t **mlpack::neighbor::RASearchRules**< SortPolicy, MetricType, TreeType >::**NumDistComputations** ( ) `[inline]`

Definition at line 215 of file ra_search_rules.hpp.

References mlpack::neighbor::RASearchRules< SortPolicy, MetricType, TreeType >::numDistComputations.

21.66.3.5 template<typename SortPolicy , typename MetricType , typename TreeType > size_t mlpack::neighbor::RASearchRules< SortPolicy, MetricType, TreeType >::NumEffectiveSamples ( ) [inline]

Definition at line 216 of file ra_search_rules.hpp.

References mlpack::neighbor::RASearchRules< SortPolicy, MetricType, TreeType >::numSamplesMade.

21.66.3.6 template<typename SortPolicy , typename MetricType , typename TreeType > void mlpack::neighbor::RASearchRules< SortPolicy, MetricType, TreeType >::ObtainDistinctSamples (const size_t *numSamples,* const size_t *rangeUpperBound,* arma::uvec & *distinctSamples*) const [private]

Pick up desired number of samples (with replacement) from a given range of integers so that only the distinct samples are returned from the range [0 - specified upper bound)

Parameters

numSamples	Number of random samples.
rangeUpperBound	The upper bound on the range of integers.
distinctSamples	The list of the distinct samples.

21.66.3.7 template<typename SortPolicy , typename MetricType , typename TreeType > double mlpack::neighbor::RASearchRules< SortPolicy, MetricType, TreeType >::Prescore (TreeType & *queryNode,* TreeType & *referenceNode,* TreeType & *referenceChildNode,* const double *baseCaseResult*) const

TOFIX: This function is specified for the cover tree (usually) so I need to think about it more algorithmically and keep its implementation mostly empty.

Also, since the access to the points in the subtree of a cover tree is non-trivial, we might have to re-work this. FOR NOW: I am just using as for a BSP-tree, I will fix it when we figure out cover trees.

21.66.3.8 template<typename SortPolicy , typename MetricType , typename TreeType > double mlpack::neighbor::RASearchRules< SortPolicy, MetricType, TreeType >::PrescoreQ (TreeType & *queryNode,* TreeType & *queryChildNode,* TreeType & *referenceNode,* const double *baseCaseResult*) const

21.66.3.9 template<typename SortPolicy , typename MetricType , typename TreeType > double mlpack::neighbor::RASearchRules< SortPolicy, MetricType, TreeType >::Rescore (const size_t *queryIndex,* TreeType & *referenceNode,* const double *oldScore*)

Re-evaluate the score for recursion order.

A low score indicates priority for recursion, while DBL_MAX indicates that the node should not be recursed into at all (it should be pruned). This is used when the score has already been calculated, but another recursion may have modified the bounds for pruning. So the old score is checked against the new pruning bound.

For rank-approximation, it also checks if the number of samples left for a query to satisfy the rank constraint is small enough at this point of the algorithm, then this node is approximated by sampling and given a new score of 'DBL_MAX'.

Parameters

queryIndex	Index of query point.
referenceNode	Candidate node to be recursed into.
oldScore	Old score produced by **Score()** (p. 346) (or **Rescore()** (p. 345)).

21.66.3.10 template<typename SortPolicy , typename MetricType , typename TreeType > double mlpack::neighbor::RASearchRules< SortPolicy, MetricType, TreeType >::Rescore (TreeType & *queryNode,* TreeType & *referenceNode,* const double *oldScore*)

Re-evaluate the score for recursion order.

A low score indicates priority for recursion, while DBL_MAX indicates that the node should not be recursed into at all (it should be pruned). This is used when the score has already been calculated, but another recursion may have modified the bounds for pruning. So the old score is checked against the new pruning bound.

For the rank-approximation, we check if the referenceNode can be approximated by sampling. If it can be, enough samples are made for every query in the queryNode. No further query-tree traversal is performed.

The 'NumSamplesMade' query stat is propagated up the tree. And then if pruning occurs (by distance or by sampling), the 'NumSamplesMade' stat is not propagated down the tree. If no pruning occurs, the stat is propagated down the tree.

Parameters

queryNode	Candidate query node to recurse into.
referenceNode	Candidate reference node to recurse into.
oldScore	Old score produced by Socre() (or **Rescore()** (p. 345)).

21.66.3.11 template<typename SortPolicy , typename MetricType , typename TreeType > double mlpack::neighbor::RASearchRules< SortPolicy, MetricType, TreeType >::Score (const size_t *queryIndex,* TreeType & *referenceNode*)

Get the score for recursion order.

A low score indicates priority for recursion, while DBL_MAX indicates that the node should not be recursed into at all (it should be pruned).

For rank-approximation, the scoring function first checks if pruning by distance is possible. If yes, then the node is given the score of 'DBL_MAX' and the expected number of samples from that node are added to the number of samples made for the query.

If no, then the function tries to see if the node can be pruned by approximation. If number of samples required from this node is small enough, then that number of samples are acquired from this node and the score is set to be 'DBL_MAX'.

If the pruning by approximation is not possible either, the algorithm continues with the usual tree-traversal.

Parameters

queryIndex	Index of query point.
referenceNode	Candidate node to be recursed into.

21.66.3.12 template<typename SortPolicy , typename MetricType , typename TreeType > double mlpack::neighbor::RASearchRules< SortPolicy, MetricType, TreeType >::Score (const size_t *queryIndex,* TreeType & *referenceNode,* const double *baseCaseResult*)

Get the score for recursion order.

A low score indicates priority for recursion, while DBL_MAX indicates that the node should not be recursed into at all (it should be pruned).

For rank-approximation, the scoring function first checks if pruning by distance is possible. If yes, then the node is given the score of 'DBL_MAX' and the expected number of samples from that node are added to the number of samples made for the query.

If no, then the function tries to see if the node can be pruned by approximation. If number of samples required from this node is small enough, then that number of samples are acquired from this node and the score is set to be 'DBL_MAX'.

If the pruning by approximation is not possible either, the algorithm continues with the usual tree-traversal.

Parameters

queryIndex	Index of query point.
referenceNode	Candidate node to be recursed into.
baseCaseResult	Result of BaseCase(queryIndex, referenceNode).

21.66.3.13 template<typename SortPolicy , typename MetricType , typename TreeType > double mlpack::neighbor::RASearchRules< SortPolicy, MetricType, TreeType >::Score (TreeType & *queryNode,* TreeType & *referenceNode*)

Get the score for recursion order.

A low score indicates priority for recursionm while DBL_MAX indicates that the node should not be recursed into at all (it should be pruned).

For the rank-approximation, we check if the referenceNode can be approximated by sampling. If it can be, enough samples are made for every query in the queryNode. No further query-tree traversal is performed.

The 'NumSamplesMade' query stat is propagated up the tree. And then if pruning occurs (by distance or by sampling), the 'NumSamplesMade' stat is not propagated down the tree. If no pruning occurs, the stat is propagated down the tree.

Parameters

queryNode	Candidate query node to recurse into.
referenceNode	Candidate reference node to recurse into.

21.66.3.14 template<typename SortPolicy , typename MetricType , typename TreeType > double mlpack::neighbor::RASearchRules< SortPolicy, MetricType, TreeType >::Score (TreeType & *queryNode,* TreeType & *referenceNode,* const double *baseCaseResult*)

Get the score for recursion order, passing the base case result (in the situation where it may be needed to calculate the recursion order).

A low score indicates priority for recursion, while DBL_MAX indicates that the node should not be recursed into at all (it should be pruned).

For the rank-approximation, we check if the referenceNode can be approximated by sampling. If it can be, enough samples are made for every query in the queryNode. No further query-tree traversal is performed.

The 'NumSamplesMade' query stat is propagated up the tree. And then if pruning occurs (by distance or by sampling), the 'NumSamplesMade' stat is not propagated down the tree. If no pruning occurs, the stat is propagated down the tree.

Parameters

queryNode	Candidate query node to recurse into.
referenceNode	Candidate reference node to recurse into.
baseCaseResult	Result of BaseCase(queryIndex, referenceNode).

21.66.3.15 template<typename SortPolicy , typename MetricType , typename TreeType > double mlpack::neighbor::RASearchRules< SortPolicy, MetricType, TreeType >::Score (const size_t *queryIndex,* TreeType & *referenceNode,* const double *distance,* const double *bestDistance*) [private]

Perform actual scoring for single-tree case.

21.66.3.16 template<typename SortPolicy , typename MetricType , typename TreeType > double mlpack::neighbor::RASearchRules< SortPolicy, MetricType, TreeType >::Score (TreeType & *queryNode,* TreeType & *referenceNode,* const double *distance,* const double *bestDistance*) [private]

Perform actual scoring for dual-tree case.

21.66.3.17 template<typename SortPolicy , typename MetricType , typename TreeType > double mlpack::neighbor::RASearchRules< SortPolicy, MetricType, TreeType >::SuccessProbability (const size_t *n,* const size_t *k,* const size_t *m,* const size_t *t*) const [private]

Compute the success probability of obtaining 'k'-neighbors from a set of size 'n' within the top 't' neighbors if 'm' samples are made.

Parameters

n	Size of the set being sampled from.
k	The number of neighbors required within the rank-approximation.
m	The number of random samples.
t	The desired rank-approximation.

21.66.4 Member Data Documentation

21.66.4.1 template<typename SortPolicy , typename MetricType , typename TreeType > arma::mat& mlpack::neighbor::RASearchRules< SortPolicy, MetricType, TreeType >::distances [private]

The matrix the resultant neighbor distances should be stored in.

Definition at line 235 of file ra_search_rules.hpp.

21.66.4.2 template<typename SortPolicy , typename MetricType , typename TreeType > bool mlpack::neighbor::RASearchRules< SortPolicy, MetricType, TreeType >::firstLeafExact [private]

Whether to do exact computation on the first leaf before any sampling.

Definition at line 244 of file ra_search_rules.hpp.

21.66.4.3 template<typename SortPolicy , typename MetricType , typename TreeType > MetricType& mlpack::neighbor::RASearchRules< SortPolicy, MetricType, TreeType >::metric [private]

The instantiated metric.

Definition at line 238 of file ra_search_rules.hpp.

21.66.4.4 template<typename SortPolicy , typename MetricType , typename TreeType > arma::Mat<size_t>& mlpack::neighbor::RASearchRules< SortPolicy, MetricType, TreeType >::neighbors [private]

The matrix the resultant neighbor indices should be stored in.

Definition at line 232 of file ra_search_rules.hpp.

21.66.4.5 template<typename SortPolicy , typename MetricType , typename TreeType > size_t mlpack::neighbor::RASearchRules< SortPolicy, MetricType, TreeType >::numDistComputations [private]

Definition at line 259 of file ra_search_rules.hpp.

Referenced by mlpack::neighbor::RASearchRules< SortPolicy, MetricType, TreeType >::NumDistComputations().

21.66.4.6 template<typename SortPolicy , typename MetricType , typename TreeType > arma::Col<size_t> mlpack::neighbor::RASearchRules< SortPolicy, MetricType, TreeType >::numSamplesMade [private]

The number of samples made for every query.

Definition at line 253 of file ra_search_rules.hpp.

Referenced by mlpack::neighbor::RASearchRules< SortPolicy, MetricType, TreeType >::NumEffectiveSamples().

21.66.4.7 template<typename SortPolicy , typename MetricType , typename TreeType > size_t mlpack::neighbor::RASearchRules< SortPolicy, MetricType, TreeType >::numSamplesReqd [private]

The minimum number of samples required per query.

Definition at line 250 of file ra_search_rules.hpp.

21.66.4.8 template<typename SortPolicy , typename MetricType , typename TreeType > const arma::mat& mlpack::neighbor::RASearchRules< SortPolicy, MetricType, TreeType >::querySet [private]

The query set.

Definition at line 229 of file ra_search_rules.hpp.

21.66.4.9 template<typename SortPolicy , typename MetricType , typename TreeType > const arma::mat& mlpack::neighbor::RASearchRules< SortPolicy, MetricType, TreeType >::referenceSet [private]

The reference set.

Definition at line 226 of file ra_search_rules.hpp.

21.66.4.10 template<typename SortPolicy , typename MetricType , typename TreeType > bool mlpack::neighbor::RASearchRules< SortPolicy, MetricType, TreeType >::sampleAtLeaves [private]

Whether to sample at leaves or just use all of it.

Definition at line 241 of file ra_search_rules.hpp.

21.66.4.11 template<typename SortPolicy , typename MetricType , typename TreeType > double mlpack::neighbor::RASearchRules< SortPolicy, MetricType, TreeType >::samplingRatio [private]

The sampling ratio.

Definition at line 256 of file ra_search_rules.hpp.

21.66.4.12 template<typename SortPolicy , typename MetricType , typename TreeType > size_t mlpack::neighbor::RASearchRules< SortPolicy, MetricType, TreeType >::singleSampleLimit [private]

The limit on the largest node that can be approximated by sampling.

Definition at line 247 of file ra_search_rules.hpp.

The documentation for this class was generated from the following file:

- src/mlpack/methods/rann/**ra_search_rules.hpp**

21.67 mlpack::nmf::HAlternatingLeastSquaresRule Class Reference

The update rule for the encoding matrix H.

Public Member Functions

- **HAlternatingLeastSquaresRule** ()

Static Public Member Functions

- template<typename MatType >
  static void **Update** (const MatType &V, const arma::mat &W, arma::mat &H)

  The update function that actually updates the H matrix.

21.67.1 Detailed Description

The update rule for the encoding matrix H.

The formula used is

$$H = \frac{W^T V}{W^T W}$$

Definition at line 82 of file als_update_rules.hpp.

21.67.2 Constructor & Destructor Documentation

21.67.2.1 mlpack::nmf::HAlternatingLeastSquaresRule::HAlternatingLeastSquaresRule ( ) [inline]

Definition at line 86 of file als_update_rules.hpp.

21.67.3 Member Function Documentation

21.67.3.1 template<typename MatType > static void mlpack::nmf::HAlternatingLeastSquaresRule::Update (const MatType & *V*, const arma::mat & *W*, arma::mat & *H*) [inline], [static]

The update function that actually updates the H matrix.

The function takes in all the matrices and only changes the value of the H matrix.

Parameters

V	Input matrix to be factorized.
W	Basis matrix.
H	Encoding matrix to be updated.

Definition at line 97 of file als_update_rules.hpp.

The documentation for this class was generated from the following file:

- src/mlpack/methods/nmf/**als_update_rules.hpp**

21.68 mlpack::nmf::HMultiplicativeDistanceRule Class Reference

The update rule for the encoding matrix H.

Public Member Functions

- **HMultiplicativeDistanceRule** ()

Static Public Member Functions

- template<typename MatType >
  static void **Update** (const MatType &V, const arma::mat &W, arma::mat &H)

  The update function that actually updates the H matrix.

21.68.1 Detailed Description

The update rule for the encoding matrix H.

The formula used is

$$H_{a\mu} \leftarrow H_{a\mu} \frac{(W^T V)_{a\mu}}{(W^T W H)_{a\mu}}$$

Definition at line 70 of file mult_dist_update_rules.hpp.

21.68.2 Constructor & Destructor Documentation

21.68.2.1 mlpack::nmf::HMultiplicativeDistanceRule::HMultiplicativeDistanceRule ( ) [inline]

Definition at line 74 of file mult_dist_update_rules.hpp.

21.68.3 Member Function Documentation

21.68.3.1 template<typename MatType > static void mlpack::nmf::HMultiplicativeDistanceRule::Update (const MatType & *V*, const arma::mat & *W*, arma::mat & *H*) [inline], [static]

The update function that actually updates the H matrix.

The function takes in all the matrices and only changes the value of the H matrix.

Parameters

V	Input matrix to be factorized.
W	Basis matrix.
H	Encoding matrix to be updated.

Definition at line 85 of file mult_dist_update_rules.hpp.

The documentation for this class was generated from the following file:

- src/mlpack/methods/nmf/**mult_dist_update_rules.hpp**

21.69 mlpack::nmf::HMultiplicativeDivergenceRule Class Reference

The update rule for the encoding matrix H.

Public Member Functions

- **HMultiplicativeDivergenceRule** ()

Static Public Member Functions

- template<typename MatType >
  static void **Update** (const MatType &V, const arma::mat &W, arma::mat &H)

  The update function that actually updates the H matrix.

21.69.1 Detailed Description

The update rule for the encoding matrix H.

The formula used is

$$H_{a\mu} \leftarrow H_{a\mu} \frac{\sum_i W_{ia} V_{i\mu} / (WH)_{i\mu}}{\sum_k H_{ka}}$$

Definition at line 94 of file mult_div_update_rules.hpp.

21.69.2 Constructor & Destructor Documentation

21.69.2.1 mlpack::nmf::HMultiplicativeDivergenceRule::HMultiplicativeDivergenceRule ( ) [inline]

Definition at line 98 of file mult_div_update_rules.hpp.

21.69.3 Member Function Documentation

21.69.3.1 template<typename MatType > static void mlpack::nmf::HMultiplicativeDivergenceRule::Update (const MatType & *V*, const arma::mat & *W*, arma::mat & *H*) [inline], [static]

The update function that actually updates the H matrix.

The function takes in all the matrices and only changes the value of the H matrix.

Parameters

V	Input matrix to be factorized.
W	Basis matrix.
H	Encoding matrix to updated.

Definition at line 109 of file mult_div_update_rules.hpp.

The documentation for this class was generated from the following file:

- src/mlpack/methods/nmf/**mult_div_update_rules.hpp**

21.70 mlpack::nmf::NMF< InitializationRule, WUpdateRule, HUpdateRule > Class Template Reference

This class implements the **NMF** (p. 353) on the given matrix V.

Public Member Functions

- **NMF** (const size_t **maxIterations**=10000, const double **minResidue**=1e-10, const InitializationRule initializeRule=InitializationRule(), const WUpdateRule wUpdate=WUpdateRule(), const HUpdateRule hUpdate=HUpdateRule())

  *Create the **NMF** (p. 353) object and (optionally) set the parameters which **NMF** (p. 353) will run with.*
- template<typename MatType >
  void **Apply** (const MatType &V, const size_t r, arma::mat &W, arma::mat &H) const

  Apply Non-Negative Matrix Factorization to the provided matrix.
- const HUpdateRule & **HUpdate** () const

  Access the H update rule.
- HUpdateRule & **HUpdate** ()

  Modify the H update rule.
- const InitializationRule & **InitializeRule** () const

  Access the initialization rule.
- InitializationRule & **InitializeRule** ()

  Modify the initialization rule.
- size_t **MaxIterations** () const

Access the maximum number of iterations.

- size_t & **MaxIterations** ()

  Modify the maximum number of iterations.

- double **MinResidue** () const

  Access the minimum residue before termination.

- double & **MinResidue** ()

  Modify the minimum residue before termination.

- const WUpdateRule & **WUpdate** () const

  Access the W update rule.

- WUpdateRule & **WUpdate** ()

  Modify the W update rule.

Private Attributes

- HUpdateRule **hUpdate**

  Instantiated H update rule.

- InitializationRule **initializeRule**

  Instantiated initialization Rule.

- size_t **maxIterations**

  The maximum number of iterations allowed before giving up.

- double **minResidue**

  The minimum residue, below which iteration is considered converged.

- WUpdateRule **wUpdate**

  Instantiated W update rule.

21.70.1 Detailed Description

template<typename InitializationRule = RandomInitialization, typename WUpdateRule = WMultiplicativeDistanceRule, typename HUpdateRule = HMultiplicativeDistanceRule>class mlpack::nmf::NMF< InitializationRule, WUpdateRule, HUpdateRule >

This class implements the **NMF** (p. 353) on the given matrix V.

Non-negative Matrix Factorization decomposes V in the form $V \approx WH$ where W is called the basis matrix and H is called the encoding matrix. V is taken to be of size n x m and the obtained W is n x r and H is r x m. The size r is called the rank of the factorization.

The implementation requires two template types; the first contains the update rule for the W matrix during each iteration and the other contains the update rule for the H matrix during each iteration. This templatization allows the user to try various update rules (including ones not supplied with MLPACK) for factorization.

A simple example of how to run **NMF** (p. 353) is shown below.

```
extern arma::mat V; // Matrix that we want to perform NMF on.
size_t r = 10; // Rank of decomposition
arma::mat W; // Basis matrix
arma::mat H; // Encoding matrix

NMF<> nmf(); // Default options
nmf.Apply(V, W, H, r);
```

For more information on non-negative matrix factorization, see the following paper:

```
@article{
  title = {{Learning the parts of objects by non-negative matrix
      factorization}},
  author = {Lee, Daniel D. and Seung, H. Sebastian},
  journal = {Nature},
  month = {Oct},
  year = {1999},
  number = {6755},
  pages = {788--791},
  publisher = {Nature Publishing Group},
  url = {http://dx.doi.org/10.1038/44565}
}
```

Template Parameters

WUpdateRule	The update rule for calculating W matrix at each iteration.
HUpdateRule	The update rule for calculating H matrix at each iteration.

See Also

WMultiplicativeDistanceRule (p. 361), **HMultiplicativeDistanceRule** (p. 351)

Definition at line 86 of file nmf.hpp.

21.70.2 Constructor & Destructor Documentation

21.70.2.1 template<typename InitializationRule = RandomInitialization, typename WUpdateRule = WMultiplicativeDistanceRule, typename HUpdateRule = HMultiplicativeDistanceRule> mlpack::nmf::NMF< InitializationRule, WUpdateRule, HUpdateRule >::NMF (const size_t *maxIterations* = 10000, const double *minResidue* = 1e-10, const InitializationRule *initializeRule* = InitializationRule(), const WUpdateRule *wUpdate* = WUpdateRule(), const HUpdateRule *hUpdate* = HUpdateRule())

Create the **NMF** (p. 353) object and (optionally) set the parameters which **NMF** (p. 353) will run with.

The minimum residue refers to the root mean square of the difference between two subsequent iterations of the product $W * H$. A low residue indicates that subsequent iterations are not producing much change in W and H. Once the residue goes below the specified minimum residue, the algorithm terminates.

Parameters

maxIterations	Maximum number of iterations allowed before giving up. A value of 0 indicates no limit.
minResidue	The minimum allowed residue before the algorithm terminates.
Initialize	Optional Initialization object for initializing the W and H matrices
WUpdate	Optional WUpdateRule object; for when the update rule for the W vector has states that it needs to store.
HUpdate	Optional HUpdateRule object; for when the update rule for the H vector has states that it needs to store.

21.70.3 Member Function Documentation

21.70.3.1 template<typename InitializationRule = RandomInitialization, typename WUpdateRule = WMultiplicativeDistanceRule, typename HUpdateRule = HMultiplicativeDistanceRule> template<typename MatType > void mlpack::nmf::NMF< InitializationRule, WUpdateRule, HUpdateRule >::Apply (const MatType & *V*, const size_t *r*, arma::mat & *W*, arma::mat & *H*) const

Apply Non-Negative Matrix Factorization to the provided matrix.

Parameters

V	Input matrix to be factorized.
W	Basis matrix to be output.
H	Encoding matrix to output.
r	Rank r of the factorization.

21.70.3.2 template<typename InitializationRule = RandomInitialization, typename WUpdateRule = WMultiplicativeDistanceRule, typename HUpdateRule = HMultiplicativeDistanceRule> const HUpdateRule& mlpack::nmf::NMF< InitializationRule, WUpdateRule, HUpdateRule >::HUpdate ( ) const [inline]

Access the H update rule.

Definition at line 158 of file nmf.hpp.

References mlpack::nmf::NMF< InitializationRule, WUpdateRule, HUpdateRule >::hUpdate.

21.70.3.3 template<typename InitializationRule = RandomInitialization, typename WUpdateRule = WMultiplicativeDistanceRule, typename HUpdateRule = HMultiplicativeDistanceRule> HUpdateRule& mlpack::nmf::NMF< InitializationRule, WUpdateRule, HUpdateRule >::HUpdate ( ) [inline]

Modify the H update rule.

Definition at line 160 of file nmf.hpp.

References mlpack::nmf::NMF< InitializationRule, WUpdateRule, HUpdateRule >::hUpdate.

21.70.3.4 template<typename InitializationRule = RandomInitialization, typename WUpdateRule = WMultiplicativeDistanceRule, typename HUpdateRule = HMultiplicativeDistanceRule> const InitializationRule& mlpack::nmf::NMF< InitializationRule, WUpdateRule, HUpdateRule >::InitializeRule ( ) const [inline]

Access the initialization rule.

Definition at line 150 of file nmf.hpp.

References mlpack::nmf::NMF< InitializationRule, WUpdateRule, HUpdateRule >::initializeRule.

21.70.3.5 template<typename InitializationRule = RandomInitialization, typename WUpdateRule = WMultiplicativeDistanceRule, typename HUpdateRule = HMultiplicativeDistanceRule> InitializationRule& mlpack::nmf::NMF< InitializationRule, WUpdateRule, HUpdateRule >::InitializeRule ( ) [inline]

Modify the initialization rule.

Definition at line 152 of file nmf.hpp.

References mlpack::nmf::NMF< InitializationRule, WUpdateRule, HUpdateRule >::initializeRule.

21.70.3.6 template<typename InitializationRule = RandomInitialization, typename WUpdateRule = WMultiplicativeDistanceRule, typename HUpdateRule = HMultiplicativeDistanceRule> size_t mlpack::nmf::NMF< InitializationRule, WUpdateRule, HUpdateRule >::MaxIterations ( ) const [inline]

Access the maximum number of iterations.

Definition at line 142 of file nmf.hpp.

References mlpack::nmf::NMF< InitializationRule, WUpdateRule, HUpdateRule >::maxIterations.

21.70.3.7 template<typename InitializationRule = RandomInitialization, typename WUpdateRule = WMultiplicativeDistanceRule, typename HUpdateRule = HMultiplicativeDistanceRule> size_t& mlpack::nmf::NMF< InitializationRule, WUpdateRule, HUpdateRule >::MaxIterations ( ) [inline]

Modify the maximum number of iterations.

Definition at line 144 of file nmf.hpp.

References mlpack::nmf::NMF< InitializationRule, WUpdateRule, HUpdateRule >::maxIterations.

21.70.3.8 template<typename InitializationRule = RandomInitialization, typename WUpdateRule = WMultiplicativeDistanceRule, typename HUpdateRule = HMultiplicativeDistanceRule> double mlpack::nmf::NMF< InitializationRule, WUpdateRule, HUpdateRule >::MinResidue ( ) const [inline]

Access the minimum residue before termination.

Definition at line 146 of file nmf.hpp.

References mlpack::nmf::NMF< InitializationRule, WUpdateRule, HUpdateRule >::minResidue.

21.70.3.9 template<typename InitializationRule = RandomInitialization, typename WUpdateRule = WMultiplicativeDistanceRule, typename HUpdateRule = HMultiplicativeDistanceRule> double& mlpack::nmf::NMF< InitializationRule, WUpdateRule, HUpdateRule >::MinResidue ( ) [inline]

Modify the minimum residue before termination.

Definition at line 148 of file nmf.hpp.

References mlpack::nmf::NMF< InitializationRule, WUpdateRule, HUpdateRule >::minResidue.

21.70.3.10 template<typename InitializationRule = RandomInitialization, typename WUpdateRule = WMultiplicativeDistanceRule, typename HUpdateRule = HMultiplicativeDistanceRule> const WUpdateRule& mlpack::nmf::NMF< InitializationRule, WUpdateRule, HUpdateRule >::WUpdate ( ) const [inline]

Access the W update rule.

Definition at line 154 of file nmf.hpp.

References mlpack::nmf::NMF< InitializationRule, WUpdateRule, HUpdateRule >::wUpdate.

21.70.3.11 template<typename InitializationRule = RandomInitialization, typename WUpdateRule = WMultiplicativeDistanceRule, typename HUpdateRule = HMultiplicativeDistanceRule> WUpdateRule& mlpack::nmf::NMF< InitializationRule, WUpdateRule, HUpdateRule >::WUpdate ( ) [inline]

Modify the W update rule.

Definition at line 156 of file nmf.hpp.

References mlpack::nmf::NMF< InitializationRule, WUpdateRule, HUpdateRule >::wUpdate.

21.70.4 Member Data Documentation

21.70.4.1 template<typename InitializationRule = RandomInitialization, typename WUpdateRule = WMultiplicativeDistanceRule, typename HUpdateRule = HMultiplicativeDistanceRule> HUpdateRule mlpack::nmf::NMF< InitializationRule, WUpdateRule, HUpdateRule >::hUpdate [private]

Instantiated H update rule.

Definition at line 138 of file nmf.hpp.

Referenced by mlpack::nmf::NMF< InitializationRule, WUpdateRule, HUpdateRule >::HUpdate().

21.70.4.2 template<typename InitializationRule = RandomInitialization, typename WUpdateRule = WMultiplicativeDistanceRule, typename HUpdateRule = HMultiplicativeDistanceRule> InitializationRule mlpack::nmf::NMF< InitializationRule, WUpdateRule, HUpdateRule >::initializeRule [private]

Instantiated initialization Rule.

Definition at line 134 of file nmf.hpp.

Referenced by mlpack::nmf::NMF< InitializationRule, WUpdateRule, HUpdateRule >::InitializeRule().

21.70.4.3 template<typename InitializationRule = RandomInitialization, typename WUpdateRule = WMultiplicativeDistanceRule, typename HUpdateRule = HMultiplicativeDistanceRule> size_t mlpack::nmf::NMF< InitializationRule, WUpdateRule, HUpdateRule >::maxIterations [private]

The maximum number of iterations allowed before giving up.

Definition at line 130 of file nmf.hpp.

Referenced by mlpack::nmf::NMF< InitializationRule, WUpdateRule, HUpdateRule >::MaxIterations().

21.70.4.4 template<typename InitializationRule = RandomInitialization, typename WUpdateRule = WMultiplicativeDistanceRule, typename HUpdateRule = HMultiplicativeDistanceRule> double mlpack::nmf::NMF< InitializationRule, WUpdateRule, HUpdateRule >::minResidue [private]

The minimum residue, below which iteration is considered converged.

Definition at line 132 of file nmf.hpp.

Referenced by mlpack::nmf::NMF< InitializationRule, WUpdateRule, HUpdateRule >::MinResidue().

21.70.4.5 template<typename InitializationRule = RandomInitialization, typename WUpdateRule = WMultiplicativeDistanceRule, typename HUpdateRule = HMultiplicativeDistanceRule> WUpdateRule mlpack::nmf::NMF< InitializationRule, WUpdateRule, HUpdateRule >::wUpdate [private]

Instantiated W update rule.

Definition at line 136 of file nmf.hpp.

Referenced by mlpack::nmf::NMF< InitializationRule, WUpdateRule, HUpdateRule >::WUpdate().

The documentation for this class was generated from the following file:

- src/mlpack/methods/nmf/**nmf.hpp**

21.71 mlpack::nmf::RandomAcolInitialization< p > Class Template Reference

This class initializes the W matrix of the **NMF** (p. 353) algorithm by averaging p randomly chosen columns of V.

Public Member Functions

- **RandomAcolInitialization** ()

Static Public Member Functions

- template<typename MatType >
  static void **Initialize** (const MatType &V, const size_t r, arma::mat &W, arma::mat &H)

21.71.1 Detailed Description

template<int p = 5>class mlpack::nmf::RandomAcolInitialization< p >

This class initializes the W matrix of the **NMF** (p. 353) algorithm by averaging p randomly chosen columns of V.

In this case, p is a template parameter. H is then set randomly.

Template Parameters

The	number of random columns to average for each column of W.

Definition at line 42 of file random_acol_init.hpp.

21.71.2 Constructor & Destructor Documentation

21.71.2.1 template<int p = 5> mlpack::nmf::RandomAcolInitialization< p >::RandomAcolInitialization ( ) [inline]

Definition at line 46 of file random_acol_init.hpp.

21.71.3 Member Function Documentation

21.71.3.1 template<int p = 5> template<typename MatType > static void mlpack::nmf::RandomAcolInitialization< p >::Initialize (const MatType & *V,* const size_t *r,* arma::mat & *W,* arma::mat & *H*) [inline], [static]

Definition at line 50 of file random_acol_init.hpp.

References mlpack::math::RandInt(), and mlpack::Log::Warn.

The documentation for this class was generated from the following file:

- src/mlpack/methods/nmf/**random_acol_init.hpp**

21.72 mlpack::nmf::RandomInitialization Class Reference

Public Member Functions

- **RandomInitialization** ()

Static Public Member Functions

- template<typename MatType >
  static void **Initialize** (const MatType &V, const size_t r, arma::mat &W, arma::mat &H)

21.72.1 Detailed Description

Definition at line 31 of file random_init.hpp.

21.72.2 Constructor & Destructor Documentation

21.72.2.1 mlpack::nmf::RandomInitialization::RandomInitialization ( ) [inline]

Definition at line 35 of file random_init.hpp.

21.72.3 Member Function Documentation

21.72.3.1 template<typename MatType > static void mlpack::nmf::RandomInitialization::Initialize (const MatType & *V,* const size_t *r,* arma::mat & *W,* arma::mat & *H*) [inline], [static]

Definition at line 38 of file random_init.hpp.

The documentation for this class was generated from the following file:

- src/mlpack/methods/nmf/**random_init.hpp**

21.73 mlpack::nmf::WAlternatingLeastSquaresRule Class Reference

The update rule for the basis matrix W.

Public Member Functions

- **WAlternatingLeastSquaresRule** ()

Static Public Member Functions

- template<typename MatType >
  static void **Update** (const MatType &V, arma::mat &W, const arma::mat &H)

  The update function that actually updates the W matrix.

21.73.1 Detailed Description

The update rule for the basis matrix W.

The formula used is

$$W^T = \frac{HV^T}{HH^T}$$

Definition at line 42 of file als_update_rules.hpp.

21.73.2 Constructor & Destructor Documentation

21.73.2.1 mlpack::nmf::WAlternatingLeastSquaresRule::WAlternatingLeastSquaresRule ( ) [inline]

Definition at line 46 of file als_update_rules.hpp.

21.73.3 Member Function Documentation

21.73.3.1 template<typename MatType > static void mlpack::nmf::WAlternatingLeastSquaresRule::Update (const MatType & *V*, arma::mat & *W*, const arma::mat & *H*) [inline], [static]

The update function that actually updates the W matrix.

The function takes in all the matrices and only changes the value of the W matrix.

Parameters

V	Input matrix to be factorized.
W	Basis matrix to be updated.
H	Encoding matrix.

Definition at line 57 of file als_update_rules.hpp.

The documentation for this class was generated from the following file:

- src/mlpack/methods/nmf/**als_update_rules.hpp**

21.74 mlpack::nmf::WMultiplicativeDistanceRule Class Reference

The update rule for the basis matrix W.

Public Member Functions

- **WMultiplicativeDistanceRule** ()

Static Public Member Functions

- template<typename MatType >
  static void **Update** (const MatType &V, arma::mat &W, const arma::mat &H)

  The update function that actually updates the W matrix.

21.74.1 Detailed Description

The update rule for the basis matrix W.

The formula used is

$$W_{ia} \leftarrow W_{ia} \frac{(VH^T)_{ia}}{(WHH^T)_{ia}}$$

Definition at line 41 of file mult_dist_update_rules.hpp.

21.74.2 Constructor & Destructor Documentation

21.74.2.1 mlpack::nmf::WMultiplicativeDistanceRule::WMultiplicativeDistanceRule ( ) [inline]

Definition at line 45 of file mult_dist_update_rules.hpp.

21.74.3 Member Function Documentation

21.74.3.1 template<typename MatType > static void mlpack::nmf::WMultiplicativeDistanceRule::Update (const MatType & *V,* arma::mat & *W,* const arma::mat & *H*) [inline], [static]

The update function that actually updates the W matrix.

The function takes in all the matrices and only changes the value of the W matrix.

Parameters

V	Input matrix to be factorized.
W	Basis matrix to be updated.
H	Encoding matrix.

Definition at line 56 of file mult_dist_update_rules.hpp.

The documentation for this class was generated from the following file:

- src/mlpack/methods/nmf/**mult_dist_update_rules.hpp**

21.75 mlpack::nmf::WMultiplicativeDivergenceRule Class Reference

The update rule for the basis matrix W.

Public Member Functions

- **WMultiplicativeDivergenceRule** ()

Static Public Member Functions

- template<typename MatType >
  static void **Update** (const MatType &V, arma::mat &W, const arma::mat &H)

  The update function that actually updates the W matrix.

21.75.1 Detailed Description

The update rule for the basis matrix W.

The formula used is

$$W_{ia} \leftarrow W_{ia} \frac{\sum_\mu H_{a\mu} V_{i\mu} / (WH)_{i\mu}}{\sum_\nu H_{a\nu}}$$

Definition at line 43 of file mult_div_update_rules.hpp.

21.75.2 Constructor & Destructor Documentation

21.75.2.1 mlpack::nmf::WMultiplicativeDivergenceRule::WMultiplicativeDivergenceRule ( ) [inline]

Definition at line 47 of file mult_div_update_rules.hpp.

21.75.3 Member Function Documentation

21.75.3.1 template<typename MatType > static void mlpack::nmf::WMultiplicativeDivergenceRule::Update (const MatType & *V*, arma::mat & *W*, const arma::mat & *H*) [inline], [static]

The update function that actually updates the W matrix.

The function takes in all the matrices and only changes the value of the W matrix.

Parameters

V	Input matrix to be factorized.
W	Basis matrix to be updated.
H	Encoding matrix.

Definition at line 58 of file mult_div_update_rules.hpp.

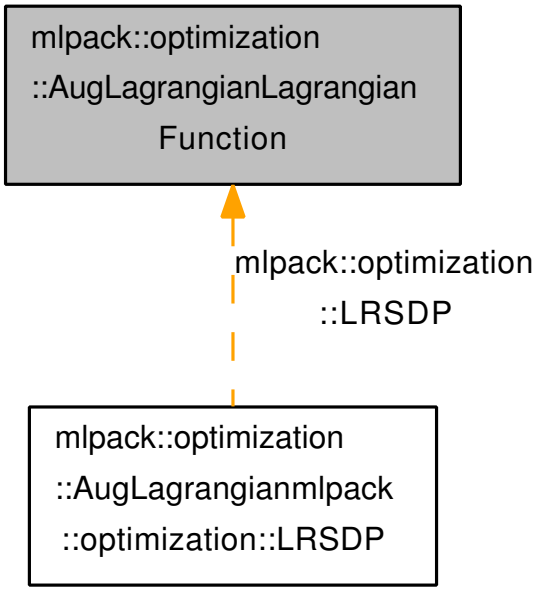
The documentation for this class was generated from the following file:

- src/mlpack/methods/nmf/**mult_div_update_rules.hpp**

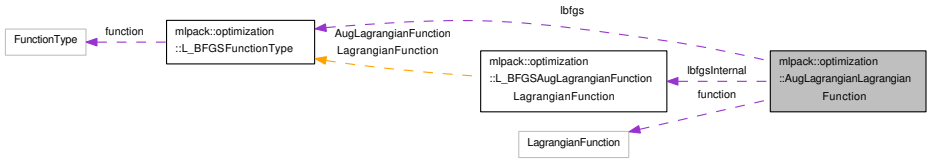
21.76 mlpack::optimization::AugLagrangian< LagrangianFunction > Class Template Reference

The **AugLagrangian** (p. 363) class implements the Augmented Lagrangian method of optimization.

Inheritance diagram for mlpack::optimization::AugLagrangian< LagrangianFunction >:



Collaboration diagram for mlpack::optimization::AugLagrangian< LagrangianFunction >:



Public Types

- typedef **L_BFGS**
  < **AugLagrangianFunction**
  < LagrangianFunction > > **L_BFGSType**

  Shorthand for the type of the L-BFGS optimizer we'll be using.

Public Member Functions

- **AugLagrangian** (LagrangianFunction &**function**)

  Initialize the Augmented Lagrangian with the default L-BFGS optimizer.
- **AugLagrangian** (**AugLagrangianFunction**< LagrangianFunction > &**augfunc**, **L_BFGSType** &**lbfgs**)

  Initialize the Augmented Lagrangian with a custom L-BFGS optimizer.
- const LagrangianFunction & **Function** () const

  Get the LagrangianFunction.
- LagrangianFunction & **Function** ()

  Modify the LagrangianFunction.
- const arma::vec & **Lambda** () const

Get the Lagrange multipliers.

- arma::vec & **Lambda** ()

  Modify the Lagrange multipliers (i.e. set them before optimization).
- const **L_BFGSType** & **LBFGS** () const

  Get the L-BFGS object used for the actual optimization.
- **L_BFGSType** & **LBFGS** ()

  Modify the L-BFGS object used for the actual optimization.
- bool **Optimize** (arma::mat &coordinates, const size_t maxIterations=1000)

  Optimize the function.
- bool **Optimize** (arma::mat &coordinates, const arma::vec &initLambda, const double initSigma, const size_t maxIterations=1000)

  Optimize the function, giving initial estimates for the Lagrange multipliers.
- double **Sigma** () const

  Get the penalty parameter.
- double & **Sigma** ()

  Modify the penalty parameter.

Private Attributes

- **AugLagrangianFunction**
  < LagrangianFunction > **augfunc**

  *Internally used **AugLagrangianFunction** (p. 369) which holds the function we are optimizing.*
- LagrangianFunction & **function**

  Function to be optimized.
- **L_BFGSType** & **lbfgs**

  The L-BFGS optimizer that we will use.
- **L_BFGSType lbfgsInternal**

  *If the user did not pass an **L_BFGS** (p. 376) object, we'll use our own internal one.*

21.76.1 Detailed Description

template<typename LagrangianFunction>class mlpack::optimization::AugLagrangian< LagrangianFunction >

The **AugLagrangian** (p. 363) class implements the Augmented Lagrangian method of optimization.

In this scheme, a penalty term is added to the Lagrangian. This method is also called the "method of multipliers".

The template class LagrangianFunction must implement the following five methods:

- double Evaluate(const arma::mat& coordinates);
- void Gradient(const arma::mat& coordinates, arma::mat& gradient);
- size_t NumConstraints();
- double EvaluateConstraint(size_t index, const arma::mat& coordinates);
- double GradientConstraint(size_t index, const arma::mat& coordinates, arma::mat& gradient);

The number of constraints must be greater than or equal to 0, and EvaluateConstraint() should evaluate the constraint at the given index for the given coordinates. Evaluate() should provide the objective function value for the given coordinates.

Template Parameters

LagrangianFunction	Function which can be optimized by this class.

Definition at line 59 of file aug_lagrangian.hpp.

21.76.2 Member Typedef Documentation

21.76.2.1 template<typename LagrangianFunction> typedef L_BFGS<AugLagrangianFunction<LagrangianFunction> > mlpack::optimization::AugLagrangian< LagrangianFunction >::L_BFGSType

Shorthand for the type of the L-BFGS optimizer we'll be using.

Definition at line 64 of file aug_lagrangian.hpp.

21.76.3 Constructor & Destructor Documentation

21.76.3.1 template<typename LagrangianFunction> mlpack::optimization::AugLagrangian< LagrangianFunction >::AugLagrangian (LagrangianFunction & *function*)

Initialize the Augmented Lagrangian with the default L-BFGS optimizer.

We limit the number of L-BFGS iterations to 1000, rather than the unlimited default L-BFGS.

Parameters

function	The function to be optimized.

21.76.3.2 template<typename LagrangianFunction> mlpack::optimization::AugLagrangian< LagrangianFunction >::AugLagrangian (AugLagrangianFunction< LagrangianFunction > & *augfunc,* L_BFGSType & *lbfgs*)

Initialize the Augmented Lagrangian with a custom L-BFGS optimizer.

Parameters

function	The function to be optimized. This must be a pre-created utility **AugLagrangianFunction** (p. 369).
lbfgs	The custom L-BFGS optimizer to be used. This should have already been initialized with the given **AugLagrangianFunction** (p. 369).

21.76.4 Member Function Documentation

21.76.4.1 template<typename LagrangianFunction> const LagrangianFunction& mlpack::optimization::AugLagrangian< LagrangianFunction >::Function ( ) const [inline]

Get the LagrangianFunction.

Definition at line 117 of file aug_lagrangian.hpp.

21.76.4.2 template<typename LagrangianFunction> LagrangianFunction& mlpack::optimization::AugLagrangian< LagrangianFunction >::Function ( ) [inline]

Modify the LagrangianFunction.

Definition at line 119 of file aug_lagrangian.hpp.

21.76.4.3 template<typename LagrangianFunction> const arma::vec& mlpack::optimization::AugLagrangian< LagrangianFunction >::Lambda ( ) const [inline]

Get the Lagrange multipliers.

Definition at line 127 of file aug_lagrangian.hpp.

21.76.4.4 template<typename LagrangianFunction> arma::vec& mlpack::optimization::AugLagrangian< LagrangianFunction >::Lambda ( ) [inline]

Modify the Lagrange multipliers (i.e. set them before optimization).

Definition at line 129 of file aug_lagrangian.hpp.

21.76.4.5 template<typename LagrangianFunction> const L_BFGSType& mlpack::optimization::AugLagrangian< LagrangianFunction >::LBFGS ( ) const [inline]

Get the L-BFGS object used for the actual optimization.

Definition at line 122 of file aug_lagrangian.hpp.

21.76.4.6 template<typename LagrangianFunction> L_BFGSType& mlpack::optimization::AugLagrangian< LagrangianFunction >::LBFGS ( ) [inline]

Modify the L-BFGS object used for the actual optimization.

Definition at line 124 of file aug_lagrangian.hpp.

21.76.4.7 template<typename LagrangianFunction> bool mlpack::optimization::AugLagrangian< LagrangianFunction >::Optimize (arma::mat & *coordinates*, const size_t *maxIterations* = 1000)

Optimize the function.

The value '1' is used for the initial value of each Lagrange multiplier. To set the Lagrange multipliers yourself, use the other overload of **Optimize()** (p. 367).

Parameters

coordinates	Output matrix to store the optimized coordinates in.
maxIterations	Maximum number of iterations of the Augmented Lagrangian algorithm. 0 indicates no maximum.
sigma	Initial penalty parameter.

21.76.4.8 template<typename LagrangianFunction> bool mlpack::optimization::AugLagrangian< LagrangianFunction >::Optimize (arma::mat & *coordinates*, const arma::vec & *initLambda*, const double *initSigma*, const size_t *maxIterations* = 1000)

Optimize the function, giving initial estimates for the Lagrange multipliers.

The vector of Lagrange multipliers will be modified to contain the Lagrange multipliers of the final solution (if one is found).

Parameters

coordinates	Output matrix to store the optimized coordinates in.
initLambda	Vector of initial Lagrange multipliers. Should have length equal to the number of constraints.
initSigma	Initial penalty parameter.
maxIterations	Maximum number of iterations of the Augmented Lagrangian algorithm. 0 indicates no maximum.

21.76.4.9 template<typename LagrangianFunction> double mlpack::optimization::AugLagrangian< LagrangianFunction >::Sigma ( ) const [inline]

Get the penalty parameter.

Definition at line 132 of file aug_lagrangian.hpp.

21.76.4.10 template<typename LagrangianFunction> double& mlpack::optimization::AugLagrangian< LagrangianFunction >::Sigma ( ) [inline]

Modify the penalty parameter.

Definition at line 134 of file aug_lagrangian.hpp.

21.76.5 Member Data Documentation

21.76.5.1 template<typename LagrangianFunction> AugLagrangianFunction<LagrangianFunction> mlpack::optimization::AugLagrangian< LagrangianFunction >::augfunc [private]

Internally used **AugLagrangianFunction** (p. 369) which holds the function we are optimizing.

This isn't publically accessible, but we provide ways to get to the Lagrange multipliers and the penalty parameter sigma.

Definition at line 143 of file aug_lagrangian.hpp.

Referenced by mlpack::optimization::AugLagrangian< mlpack::optimization::LRSDP >::Lambda(), and mlpack::optimization::AugLagrangian< mlpack::optimization::LRSDP >::Sigma().

21.76.5.2 template<typename LagrangianFunction> LagrangianFunction& mlpack::optimization::AugLagrangian< LagrangianFunction >::function [private]

Function to be optimized.

Definition at line 138 of file aug_lagrangian.hpp.

21.76.5.3 template<typename LagrangianFunction> L_BFGSType& mlpack::optimization::AugLagrangian< LagrangianFunction >::lbfgs [private]

The L-BFGS optimizer that we will use.

Definition at line 149 of file aug_lagrangian.hpp.

Referenced by mlpack::optimization::AugLagrangian< mlpack::optimization::LRSDP >::LBFGS().

21.76.5.4 template<typename LagrangianFunction> L_BFGSType mlpack::optimization::AugLagrangian< LagrangianFunction >::lbfgsInternal [private]

If the user did not pass an **L_BFGS** (p. 376) object, we'll use our own internal one.

Definition at line 146 of file aug_lagrangian.hpp.

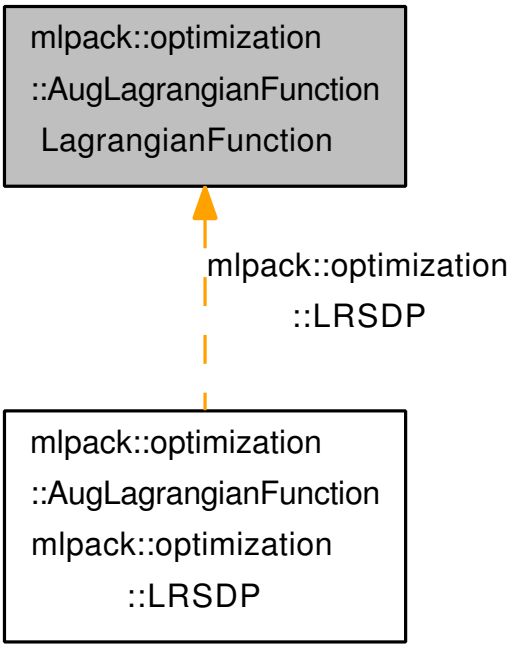
The documentation for this class was generated from the following file:

- src/mlpack/core/optimizers/aug_lagrangian/**aug_lagrangian.hpp**

21.77 mlpack::optimization::AugLagrangianFunction< LagrangianFunction > Class Template Reference

This is a utility class used by **AugLagrangian** (p. 363), meant to wrap a LagrangianFunction into a function usable by a simple optimizer like L-BFGS.

Inheritance diagram for mlpack::optimization::AugLagrangianFunction< LagrangianFunction >:



Public Member Functions

- **AugLagrangianFunction** (LagrangianFunction &**function**)

  *Initialize the **AugLagrangianFunction** (p. 369), but don't set the Lagrange multipliers or penalty parameters yet.*
- **AugLagrangianFunction** (LagrangianFunction &**function**, const arma::vec &**lambda**, const double **sigma**)

  *Initialize the **AugLagrangianFunction** (p. 369) with the given LagrangianFunction, Lagrange multipliers, and initial penalty parameter.*
- double **Evaluate** (const arma::mat &coordinates) const

  Evaluate the objective function of the Augmented Lagrangian function, which is the standard Lagrangian function evaluation plus a penalty term, which penalizes unsatisfied constraints.

- const LagrangianFunction & **Function** () const

  Get the Lagrangian function.
- LagrangianFunction & **Function** ()

  Modify the Lagrangian function.
- const arma::mat & **GetInitialPoint** () const

  Get the initial point of the optimization (supplied by the LagrangianFunction).
- void **Gradient** (const arma::mat &coordinates, arma::mat &gradient) const

  Evaluate the gradient of the Augmented Lagrangian function.
- const arma::vec & **Lambda** () const

  Get the Lagrange multipliers.
- arma::vec & **Lambda** ()

  Modify the Lagrange multipliers.
- double **Sigma** () const

  Get sigma (the penalty parameter).
- double & **Sigma** ()

  Modify sigma (the penalty parameter).

Private Attributes

- LagrangianFunction & **function**

  Instantiation of the function to be optimized.
- arma::vec **lambda**

  The Lagrange multipliers.
- double **sigma**

  The penalty parameter.

21.77.1 Detailed Description

template<typename LagrangianFunction>class mlpack::optimization::AugLagrangianFunction< LagrangianFunction >

This is a utility class used by **AugLagrangian** (p. 363), meant to wrap a LagrangianFunction into a function usable by a simple optimizer like L-BFGS.

Given a LagrangianFunction which follows the format outlined in the documentation for **AugLagrangian** (p. 363), this class provides **Evaluate()** (p. 371), **Gradient()** (p. 372), and **GetInitialPoint()** (p. 372) functions which allow this class to be used with a simple optimizer like L-BFGS.

This class can be specialized for your particular implementation – commonly, a faster method for computing the overall objective and gradient of the augmented Lagrangian function can be implemented than the naive, default implementation given. Use class template specialization and re-implement all of the methods (unfortunately, C++ specialization rules mean you have to re-implement everything).

Template Parameters

LagrangianFunction	Lagrangian function to be used.

Definition at line 48 of file aug_lagrangian_function.hpp.

21.77.2 Constructor & Destructor Documentation

21.77.2.1 template<typename LagrangianFunction> mlpack::optimization::AugLagrangianFunction< LagrangianFunction >::AugLagrangianFunction (LagrangianFunction & *function*)

Initialize the **AugLagrangianFunction** (p. 369), but don't set the Lagrange multipliers or penalty parameters yet.

Make sure you set the Lagrange multipliers before you use this...

Parameters

function	Lagrangian function.

21.77.2.2 template<typename LagrangianFunction> mlpack::optimization::AugLagrangianFunction< LagrangianFunction >::AugLagrangianFunction (LagrangianFunction & *function,* const arma::vec & *lambda,* const double *sigma*)

Initialize the **AugLagrangianFunction** (p. 369) with the given LagrangianFunction, Lagrange multipliers, and initial penalty parameter.

Parameters

function	Lagrangian function.
lambda	Initial Lagrange multipliers.
sigma	Initial penalty parameter.

21.77.3 Member Function Documentation

21.77.3.1 template<typename LagrangianFunction> double mlpack::optimization::AugLagrangianFunction< LagrangianFunction >::Evaluate (const arma::mat & *coordinates*) const

Evaluate the objective function of the Augmented Lagrangian function, which is the standard Lagrangian function evaluation plus a penalty term, which penalizes unsatisfied constraints.

Parameters

coordinates	Coordinates to evaluate function at.

Returns

Objective function.

21.77.3.2 template<typename LagrangianFunction> const LagrangianFunction& mlpack::optimization::AugLagrangianFunction< LagrangianFunction >::Function ( ) const [inline]

Get the Lagrangian function.

Definition at line 108 of file aug_lagrangian_function.hpp.

21.77.3.3 template<typename LagrangianFunction> LagrangianFunction& mlpack::optimization::AugLagrangianFunction< LagrangianFunction >::Function ( ) [inline]

Modify the Lagrangian function.

Definition at line 110 of file aug_lagrangian_function.hpp.

21.77.3.4 template<typename LagrangianFunction> const arma::mat& mlpack::optimization::AugLagrangianFunction< LagrangianFunction >::GetInitialPoint ( ) const

Get the initial point of the optimization (supplied by the LagrangianFunction).

Returns

Initial point.

21.77.3.5 template<typename LagrangianFunction> void mlpack::optimization::AugLagrangianFunction< LagrangianFunction >::Gradient (const arma::mat & *coordinates*, arma::mat & *gradient*) const

Evaluate the gradient of the Augmented Lagrangian function.

Parameters

coordinates	Coordinates to evaluate gradient at.
gradient	Matrix to store gradient into.

21.77.3.6 template<typename LagrangianFunction> const arma::vec& mlpack::optimization::AugLagrangianFunction< LagrangianFunction >::Lambda ( ) const [inline]

Get the Lagrange multipliers.

Definition at line 98 of file aug_lagrangian_function.hpp.

21.77.3.7 template<typename LagrangianFunction> arma::vec& mlpack::optimization::AugLagrangianFunction< LagrangianFunction >::Lambda ( ) [inline]

Modify the Lagrange multipliers.

Definition at line 100 of file aug_lagrangian_function.hpp.

21.77.3.8 template<typename LagrangianFunction> double mlpack::optimization::AugLagrangianFunction< LagrangianFunction >::Sigma ( ) const [inline]

Get sigma (the penalty parameter).

Definition at line 103 of file aug_lagrangian_function.hpp.

21.77.3.9 template<typename LagrangianFunction> double& mlpack::optimization::AugLagrangianFunction< LagrangianFunction >::Sigma ( ) [inline]

Modify sigma (the penalty parameter).

Definition at line 105 of file aug_lagrangian_function.hpp.

21.77.4 Member Data Documentation

21.77.4.1 template<typename LagrangianFunction> LagrangianFunction& mlpack::optimization::AugLagrangianFunction< LagrangianFunction >::function [private]

Instantiation of the function to be optimized.

Definition at line 114 of file aug_lagrangian_function.hpp.

21.77.4.2 template<typename LagrangianFunction> arma::vec mlpack::optimization::AugLagrangianFunction< LagrangianFunction >::lambda [private]

The Lagrange multipliers.

Definition at line 117 of file aug_lagrangian_function.hpp.

Referenced by mlpack::optimization::AugLagrangianFunction< mlpack::optimization::LRSDP >::Lambda().

21.77.4.3 template<typename LagrangianFunction> double mlpack::optimization::AugLagrangianFunction< LagrangianFunction >::sigma [private]

The penalty parameter.

Definition at line 119 of file aug_lagrangian_function.hpp.

Referenced by mlpack::optimization::AugLagrangianFunction< mlpack::optimization::LRSDP >::Sigma().

The documentation for this class was generated from the following file:

- src/mlpack/core/optimizers/aug_lagrangian/**aug_lagrangian_function.hpp**

21.78 mlpack::optimization::AugLagrangianTestFunction Class Reference

This function is taken from "Practical Mathematical Optimization" (Snyman), section 5.3.8 ("Application of the Augmented Lagrangian Method").

Public Member Functions

- **AugLagrangianTestFunction** ()
- **AugLagrangianTestFunction** (const arma::mat &initial_point)
- double **Evaluate** (const arma::mat &coordinates)
- double **EvaluateConstraint** (const size_t index, const arma::mat &coordinates)
- const arma::mat & **GetInitialPoint** () const
- void **Gradient** (const arma::mat &coordinates, arma::mat &gradient)
- void **GradientConstraint** (const size_t index, const arma::mat &coordinates, arma::mat &gradient)
- size_t **NumConstraints** () const

Private Attributes

- arma::mat **initialPoint**

21.78.1 Detailed Description

This function is taken from "Practical Mathematical Optimization" (Snyman), section 5.3.8 ("Application of the Augmented Lagrangian Method").

It has only one constraint.

The minimum that satisfies the constraint is x = [1, 4], with an objective value of 70.

Definition at line 38 of file aug_lagrangian_test_functions.hpp.

21.78.2 Constructor & Destructor Documentation

21.78.2.1 mlpack::optimization::AugLagrangianTestFunction::AugLagrangianTestFunction ( )

21.78.2.2 mlpack::optimization::AugLagrangianTestFunction::AugLagrangianTestFunction (const arma::mat & *initial_point*)

21.78.3 Member Function Documentation

21.78.3.1 double mlpack::optimization::AugLagrangianTestFunction::Evaluate (const arma::mat & *coordinates*)

21.78.3.2 double mlpack::optimization::AugLagrangianTestFunction::EvaluateConstraint (const size_t *index,* const arma::mat & *coordinates*)

21.78.3.3 const arma::mat& mlpack::optimization::AugLagrangianTestFunction::GetInitialPoint ( ) const [inline]

Definition at line 54 of file aug_lagrangian_test_functions.hpp.

References initialPoint.

21.78.3.4 void mlpack::optimization::AugLagrangianTestFunction::Gradient (const arma::mat & *coordinates,* arma::mat & *gradient*)

21.78.3.5 void mlpack::optimization::AugLagrangianTestFunction::GradientConstraint (const size_t *index,* const arma::mat & *coordinates,* arma::mat & *gradient*)

21.78.3.6 size_t mlpack::optimization::AugLagrangianTestFunction::NumConstraints ( ) const [inline]

Definition at line 47 of file aug_lagrangian_test_functions.hpp.

21.78.4 Member Data Documentation

21.78.4.1 arma::mat mlpack::optimization::AugLagrangianTestFunction::initialPoint [private]

Definition at line 57 of file aug_lagrangian_test_functions.hpp.

Referenced by GetInitialPoint().

The documentation for this class was generated from the following file:

- src/mlpack/core/optimizers/aug_lagrangian/**aug_lagrangian_test_functions.hpp**

21.79 mlpack::optimization::GockenbachFunction Class Reference

This function is taken from M.

Public Member Functions

- **GockenbachFunction** ()
- **GockenbachFunction** (const arma::mat &initial_point)
- double **Evaluate** (const arma::mat &coordinates)
- double **EvaluateConstraint** (const size_t index, const arma::mat &coordinates)
- const arma::mat & **GetInitialPoint** () const
- void **Gradient** (const arma::mat &coordinates, arma::mat &gradient)
- void **GradientConstraint** (const size_t index, const arma::mat &coordinates, arma::mat &gradient)
- size_t **NumConstraints** () const

Private Attributes

- arma::mat **initialPoint**

21.79.1 Detailed Description

This function is taken from M.

Gockenbach's lectures on general nonlinear programs, found at: `http://www.math.mtu.edu/~msgocken/ma5630spring20 pdf`

The program we are using is example 2.5 from this document. I have arbitrarily decided that this will be called the Gockenbach function.

The minimum that satisfies the two constraints is given as x = [0.12288, -1.1078, 0.015100], with an objective value of about 29.634.

Definition at line 71 of file aug_lagrangian_test_functions.hpp.

21.79.2 Constructor & Destructor Documentation

21.79.2.1 mlpack::optimization::GockenbachFunction::GockenbachFunction ( )

21.79.2.2 mlpack::optimization::GockenbachFunction::GockenbachFunction (const arma::mat & *initial_point*)

21.79.3 Member Function Documentation

21.79.3.1 double mlpack::optimization::GockenbachFunction::Evaluate (const arma::mat & *coordinates*)

21.79.3.2 double mlpack::optimization::GockenbachFunction::EvaluateConstraint (const size_t *index,* const arma::mat & *coordinates*)

21.79.3.3 const arma::mat& mlpack::optimization::GockenbachFunction::GetInitialPoint ( ) const [inline]

Definition at line 87 of file aug_lagrangian_test_functions.hpp.

References initialPoint.

21.79.3.4 void mlpack::optimization::GockenbachFunction::Gradient (const arma::mat & *coordinates,* arma::mat & *gradient*)

21.79.3.5 void mlpack::optimization::GockenbachFunction::GradientConstraint (const size_t *index,* const arma::mat & *coordinates,* arma::mat & *gradient*)

21.79.3.6 size_t mlpack::optimization::GockenbachFunction::NumConstraints ( ) const [inline]

Definition at line 80 of file aug_lagrangian_test_functions.hpp.

21.79.4 Member Data Documentation

21.79.4.1 arma::mat mlpack::optimization::GockenbachFunction::initialPoint [private]

Definition at line 90 of file aug_lagrangian_test_functions.hpp.

Referenced by GetInitialPoint().

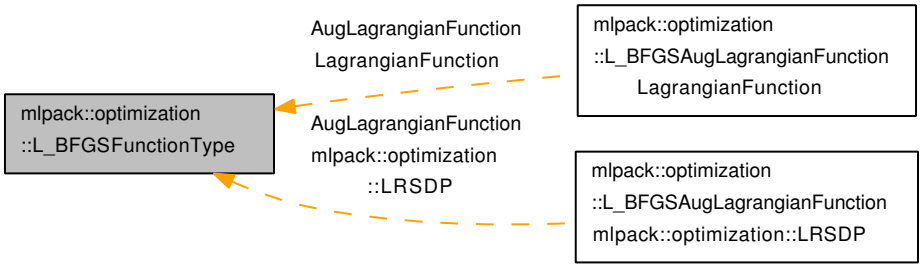
The documentation for this class was generated from the following file:

- src/mlpack/core/optimizers/aug_lagrangian/**aug_lagrangian_test_functions.hpp**

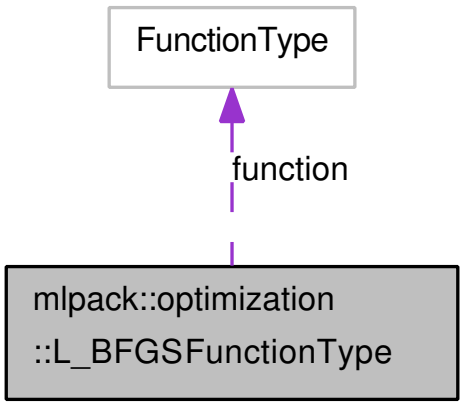
21.80 mlpack::optimization::L_BFGS< FunctionType > Class Template Reference

The generic L-BFGS optimizer, which uses a back-tracking line search algorithm to minimize a function.

Inheritance diagram for mlpack::optimization::L_BFGS< FunctionType >:



Collaboration diagram for mlpack::optimization::L_BFGS< FunctionType >:



Public Member Functions

- **L_BFGS** (FunctionType &**function**, const size_t **numBasis**=5, const size_t **maxIterations**=0, const double **armijoConstant**=1e-4, const double wolfe=0.9, const double minGradientNorm=1e-10, const size_t maxLineSearchTrials=50, const double minStep=1e-20, const double maxStep=1e20)

  Initialize the L-BFGS object.
- double **ArmijoConstant** () const

  Get the Armijo condition constant.
- double & **ArmijoConstant** ()

  Modify the Armijo condition constant.
- const FunctionType & **Function** () const

  Return the function that is being optimized.
- FunctionType & **Function** ()

  Modify the function that is being optimized.
- size_t **MaxIterations** () const

  Get the maximum number of iterations.
- size_t & **MaxIterations** ()

  Modify the maximum number of iterations.
- size_t **MaxLineSearchTrials** () const

  Get the maximum number of line search trials.
- size_t & **MaxLineSearchTrials** ()

  Modify the maximum number of line search trials.
- double **MaxStep** () const

  Return the maximum line search step size.
- double & **MaxStep** ()

  Modify the maximum line search step size.
- double **MinGradientNorm** () const

  Get the minimum gradient norm.
- double & **MinGradientNorm** ()

  Modify the minimum gradient norm.
- const std::pair< arma::mat, double > & **MinPointIterate** () const

Return the point where the lowest function value has been found.

- double **MinStep** () const

  Return the minimum line search step size.

- double & **MinStep** ()

  Modify the minimum line search step size.

- size_t **NumBasis** () const

  Get the memory size.

- size_t & **NumBasis** ()

  Modify the memory size.

- double **Optimize** (arma::mat &iterate)

  Use L-BFGS to optimize the given function, starting at the given iterate point and finding the minimum.

- double **Optimize** (arma::mat &iterate, const size_t **maxIterations**)

  Use L-BFGS to optimize (minimize) the given function, starting at the given iterate point, and performing no more than the given maximum number of iterations (the class variable maxIterations is ignored for this run, but not modified).

- double **Wolfe** () const

  Get the Wolfe parameter.

- double & **Wolfe** ()

  Modify the Wolfe parameter.

Private Member Functions

- double **ChooseScalingFactor** (const size_t iterationNum, const arma::mat &gradient)

  Calculate the scaling factor, gamma, which is used to scale the Hessian approximation matrix.

- double **Evaluate** (const arma::mat &iterate)

  Evaluate the function at the given iterate point and store the result if it is a new minimum.

- bool **GradientNormTooSmall** (const arma::mat &gradient)

  Check to make sure that the norm of the gradient is not smaller than 1e-5.

- bool **LineSearch** (double &functionValue, arma::mat &iterate, arma::mat &gradient, const arma::mat &searchDirection)

  Perform a back-tracking line search along the search direction to calculate a step size satisfying the Wolfe conditions.

- void **SearchDirection** (const arma::mat &gradient, const size_t iterationNum, const double scalingFactor, arma::mat &searchDirection)

  Find the L-BFGS search direction.

- void **UpdateBasisSet** (const size_t iterationNum, const arma::mat &iterate, const arma::mat &oldIterate, const arma::mat &gradient, const arma::mat &oldGradient)

  Update the y and s matrices, which store the differences between the iterate and old iterate and the differences between the gradient and the old gradient, respectively.

Private Attributes

- double **armijoConstant**

  Parameter for determining the Armijo condition.

- FunctionType & **function**

  Internal reference to the function we are optimizing.

- size_t **maxIterations**

  Maximum number of iterations.

- size_t **maxLineSearchTrials**

Maximum number of trials for the line search.

- double **maxStep**

  Maximum step of the line search.

- double **minGradientNorm**

  Minimum gradient norm required to continue the optimization.

- std::pair< arma::mat, double > **minPointIterate**

  Best point found so far.

- double **minStep**

  Minimum step of the line search.

- arma::mat **newIterateTmp**

  Position of the new iterate.

- size_t **numBasis**

  Size of memory for this L-BFGS optimizer.

- arma::cube **s**

  Stores all the s matrices in memory.

- double **wolfe**

  Parameter for detecting the Wolfe condition.

- arma::cube **y**

  Stores all the y matrices in memory.

21.80.1 Detailed Description

template<typename FunctionType>class mlpack::optimization::L_BFGS< FunctionType >

The generic L-BFGS optimizer, which uses a back-tracking line search algorithm to minimize a function.

The parameters for the algorithm (number of memory points, maximum step size, and so forth) are all configurable via either the constructor or standalone modifier functions. A function which can be optimized by this class must implement the following methods:

- a default constructor
- double **Evaluate(const arma::mat& coordinates)** (p. 380);
- void Gradient(const arma::mat& coordinates, arma::mat& gradient);
- arma::mat& GetInitialPoint();

Definition at line 44 of file lbfgs.hpp.

21.80.2 Constructor & Destructor Documentation

21.80.2.1 template<typename FunctionType> mlpack::optimization::L_BFGS< FunctionType >::L_BFGS (FunctionType & *function*, const size_t *numBasis* = 5, const size_t *maxIterations* = 0, const double *armijoConstant* = 1e-4, const double *wolfe* = 0.9, const double *minGradientNorm* = 1e-10, const size_t *maxLineSearchTrials* = 50, const double *minStep* = 1e-20, const double *maxStep* = 1e20)

Initialize the L-BFGS object.

Store a reference to the function we will be optimizing and set the size of the memory for the algorithm. There are many parameters that can be set for the optimization, but default values are given for each of them.

Parameters

function	Instance of function to be optimized.
numBasis	Number of memory points to be stored (default 5).
maxIterations	Maximum number of iterations for the optimization (default 0 – may run indefinitely).
armijoConstant	Controls the accuracy of the line search routine for determining the Armijo condition.
wolfe	Parameter for detecting the Wolfe condition.
minGradient-Norm	Minimum gradient norm required to continue the optimization.
maxLineSearch-Trials	The maximum number of trials for the line search (before giving up).
minStep	The minimum step of the line search.
maxStep	The maximum step of the line search.

21.80.3 Member Function Documentation

21.80.3.1 template<typename FunctionType> double mlpack::optimization::L_BFGS< FunctionType >::ArmijoConstant ( ) const [inline]

Get the Armijo condition constant.

Definition at line 128 of file lbfgs.hpp.

21.80.3.2 template<typename FunctionType> double& mlpack::optimization::L_BFGS< FunctionType >::ArmijoConstant ( ) [inline]

Modify the Armijo condition constant.

Definition at line 130 of file lbfgs.hpp.

21.80.3.3 template<typename FunctionType> double mlpack::optimization::L_BFGS< FunctionType >::ChooseScalingFactor (const size_t *iterationNum,* const arma::mat & *gradient*) [private]

Calculate the scaling factor, gamma, which is used to scale the Hessian approximation matrix.

See method M3 in Section 4 of Liu and Nocedal (1989).

Returns

The calculated scaling factor.

21.80.3.4 template<typename FunctionType> double mlpack::optimization::L_BFGS< FunctionType >::Evaluate (const arma::mat & *iterate*) [private]

Evaluate the function at the given iterate point and store the result if it is a new minimum.

Returns

The value of the function.

21.80.3.5 template<typename FunctionType> const FunctionType& mlpack::optimization::L_BFGS< FunctionType >::Function ( ) const [inline]

Return the function that is being optimized.

Definition at line 113 of file lbfgs.hpp.

21.80.3.6 template<typename FunctionType> FunctionType& mlpack::optimization::L_BFGS< FunctionType >::Function ( ) [inline]

Modify the function that is being optimized.

Definition at line 115 of file lbfgs.hpp.

21.80.3.7 template<typename FunctionType> bool mlpack::optimization::L_BFGS< FunctionType >::GradientNormTooSmall (const arma::mat & *gradient*) [private]

Check to make sure that the norm of the gradient is not smaller than 1e-5.

Currently that value is not configurable.

Returns

(norm < minGradientNorm).

21.80.3.8 template<typename FunctionType> bool mlpack::optimization::L_BFGS< FunctionType >::LineSearch (double & *functionValue,* arma::mat & *iterate,* arma::mat & *gradient,* const arma::mat & *searchDirection*) [private]

Perform a back-tracking line search along the search direction to calculate a step size satisfying the Wolfe conditions.

The parameter iterate will be modified if the method is successful.

Parameters

functionValue	Value of the function at the initial point
iterate	The initial point to begin the line search from
gradient	The gradient at the initial point
searchDirection	A vector specifying the search direction
stepSize	Variable the calculated step size will be stored in

Returns

false if no step size is suitable, true otherwise.

21.80.3.9 template<typename FunctionType> size_t mlpack::optimization::L_BFGS< FunctionType >::MaxIterations ( ) const [inline]

Get the maximum number of iterations.

Definition at line 123 of file lbfgs.hpp.

21.80.3.10 template<typename FunctionType> size_t& mlpack::optimization::L_BFGS< FunctionType >::MaxIterations ( ) [inline]

Modify the maximum number of iterations.

Definition at line 125 of file lbfgs.hpp.

21.80.3.11 template<typename FunctionType> size_t mlpack::optimization::L_BFGS< FunctionType >::MaxLineSearchTrials ( ) const [inline]

Get the maximum number of line search trials.

Definition at line 143 of file lbfgs.hpp.

21.80.3.12 template<typename FunctionType> size_t& mlpack::optimization::L_BFGS< FunctionType >::MaxLineSearchTrials ( ) [inline]

Modify the maximum number of line search trials.

Definition at line 145 of file lbfgs.hpp.

21.80.3.13 template<typename FunctionType> double mlpack::optimization::L_BFGS< FunctionType >::MaxStep ( ) const [inline]

Return the maximum line search step size.

Definition at line 153 of file lbfgs.hpp.

21.80.3.14 template<typename FunctionType> double& mlpack::optimization::L_BFGS< FunctionType >::MaxStep ( ) [inline]

Modify the maximum line search step size.

Definition at line 155 of file lbfgs.hpp.

21.80.3.15 template<typename FunctionType> double mlpack::optimization::L_BFGS< FunctionType >::MinGradientNorm ( ) const [inline]

Get the minimum gradient norm.

Definition at line 138 of file lbfgs.hpp.

21.80.3.16 template<typename FunctionType> double& mlpack::optimization::L_BFGS< FunctionType >::MinGradientNorm ( ) [inline]

Modify the minimum gradient norm.

Definition at line 140 of file lbfgs.hpp.

21.80.3.17 template<typename FunctionType> const std::pair<arma::mat, double>& mlpack::optimization::L_BFGS< FunctionType >::MinPointIterate ( ) const

Return the point where the lowest function value has been found.

Returns

arma::vec representing the point and a double with the function value at that point.

21.80.3.18 template<typename FunctionType> double mlpack::optimization::L_BFGS< FunctionType >::MinStep ( ) const [inline]

Return the minimum line search step size.

Definition at line 148 of file lbfgs.hpp.

21.80.3.19 template<typename FunctionType> double& mlpack::optimization::L_BFGS< FunctionType >::MinStep ( ) [inline]

Modify the minimum line search step size.

Definition at line 150 of file lbfgs.hpp.

21.80.3.20 template<typename FunctionType> size_t mlpack::optimization::L_BFGS< FunctionType >::NumBasis ( ) const [inline]

Get the memory size.

Definition at line 118 of file lbfgs.hpp.

21.80.3.21 template<typename FunctionType> size_t& mlpack::optimization::L_BFGS< FunctionType >::NumBasis ( ) [inline]

Modify the memory size.

Definition at line 120 of file lbfgs.hpp.

21.80.3.22 template<typename FunctionType> double mlpack::optimization::L_BFGS< FunctionType >::Optimize (arma::mat & *iterate*)

Use L-BFGS to optimize the given function, starting at the given iterate point and finding the minimum.

The maximum number of iterations is set in the constructor (or with **MaxIterations()** (p. 382)). Alternately, another overload is provided which takes a maximum number of iterations as a parameter. The given starting point will be modified to store the finishing point of the algorithm, and the final objective value is returned.

Parameters

iterate	Starting point (will be modified).

Returns

Objective value of the final point.

21.80.3.23 template<typename FunctionType> double mlpack::optimization::L_BFGS< FunctionType >::Optimize (arma::mat & *iterate,* const size_t *maxIterations*)

Use L-BFGS to optimize (minimize) the given function, starting at the given iterate point, and performing no more than the given maximum number of iterations (the class variable maxIterations is ignored for this run, but not modified).

The given starting point will be modified to store the finishing point of the algorithm, and the final objective value is returned.

Parameters

iterate	Starting point (will be modified).
maxIterations	Maximum number of iterations (0 specifies no limit).

Returns

Objective value of the final point.

21.80.3.24 template<typename FunctionType> void mlpack::optimization::L_BFGS< FunctionType >::SearchDirection (const arma::mat & *gradient,* const size_t *iterationNum,* const double *scalingFactor,* arma::mat & *searchDirection*) [private]

Find the L-BFGS search direction.

Parameters

gradient	The gradient at the current point
iteration_num	The iteration number
scaling_factor	Scaling factor to use (see ChooseScalingFactor_())
search_direction	Vector to store search direction in

21.80.3.25 template<typename FunctionType> void mlpack::optimization::L_BFGS< FunctionType >::UpdateBasisSet (const size_t *iterationNum,* const arma::mat & *iterate,* const arma::mat & *oldIterate,* const arma::mat & *gradient,* const arma::mat & *oldGradient*) [private]

Update the y and s matrices, which store the differences between the iterate and old iterate and the differences between the gradient and the old gradient, respectively.

Parameters

iterationNum	Iteration number
iterate	Current point
oldIterate	Point at last iteration
gradient	Gradient at current point (iterate)
oldGradient	Gradient at last iteration point (oldIterate)

21.80.3.26 template<typename FunctionType> double mlpack::optimization::L_BFGS< FunctionType >::Wolfe ( ) const [inline]

Get the Wolfe parameter.

Definition at line 133 of file lbfgs.hpp.

21.80.3.27 template<typename FunctionType> double& mlpack::optimization::L_BFGS< FunctionType >::Wolfe ( ) [inline]

Modify the Wolfe parameter.

Definition at line 135 of file lbfgs.hpp.

21.80.4 Member Data Documentation

21.80.4.1 template<typename FunctionType> double mlpack::optimization::L_BFGS< FunctionType >::armijoConstant [private]

Parameter for determining the Armijo condition.

Definition at line 173 of file lbfgs.hpp.

Referenced by mlpack::optimization::L_BFGS< AugLagrangianFunction< mlpack::optimization::LRSDP > >::ArmijoConstant().

21.80.4.2 template<typename FunctionType> FunctionType& mlpack::optimization::L_BFGS< FunctionType >::function [private]

Internal reference to the function we are optimizing.

Definition at line 159 of file lbfgs.hpp.

21.80.4.3 template<typename FunctionType> size_t mlpack::optimization::L_BFGS< FunctionType >::maxIterations [private]

Maximum number of iterations.

Definition at line 171 of file lbfgs.hpp.

Referenced by mlpack::optimization::L_BFGS< AugLagrangianFunction< mlpack::optimization::LRSDP > >::MaxIterations().

21.80.4.4 template<typename FunctionType> size_t mlpack::optimization::L_BFGS< FunctionType >::maxLineSearchTrials [private]

Maximum number of trials for the line search.

Definition at line 179 of file lbfgs.hpp.

Referenced by mlpack::optimization::L_BFGS< AugLagrangianFunction< mlpack::optimization::LRSDP > >::MaxLineSearchTrials().

21.80.4.5 template<typename FunctionType> double mlpack::optimization::L_BFGS< FunctionType >::maxStep [private]

Maximum step of the line search.

Definition at line 183 of file lbfgs.hpp.

Referenced by mlpack::optimization::L_BFGS< AugLagrangianFunction< mlpack::optimization::LRSDP > >::MaxStep().

21.80.4.6 template<typename FunctionType> double mlpack::optimization::L_BFGS< FunctionType >::minGradientNorm [private]

Minimum gradient norm required to continue the optimization.

Definition at line 177 of file lbfgs.hpp.

Referenced by mlpack::optimization::L_BFGS< AugLagrangianFunction< mlpack::optimization::LRSDP > >::MinGradientNorm().

21.80.4.7 template<typename FunctionType> std::pair<arma::mat, double> mlpack::optimization::L_BFGS< FunctionType >::minPointIterate [private]

Best point found so far.

Definition at line 186 of file lbfgs.hpp.

21.80.4.8 template<typename FunctionType> double mlpack::optimization::L_BFGS< FunctionType >::minStep [private]

Minimum step of the line search.

Definition at line 181 of file lbfgs.hpp.

Referenced by mlpack::optimization::L_BFGS< AugLagrangianFunction< mlpack::optimization::LRSDP > >::MinStep().

21.80.4.9 template<typename FunctionType> arma::mat mlpack::optimization::L_BFGS< FunctionType >::newIterateTmp [private]

Position of the new iterate.

Definition at line 162 of file lbfgs.hpp.

21.80.4.10 template<typename FunctionType> size_t mlpack::optimization::L_BFGS< FunctionType >::numBasis [private]

Size of memory for this L-BFGS optimizer.

Definition at line 169 of file lbfgs.hpp.

Referenced by mlpack::optimization::L_BFGS< AugLagrangianFunction< mlpack::optimization::LRSDP > >::NumBasis().

21.80.4.11 template<typename FunctionType> arma::cube mlpack::optimization::L_BFGS< FunctionType >::s [private]

Stores all the s matrices in memory.

Definition at line 164 of file lbfgs.hpp.

21.80.4.12 template<typename FunctionType> double mlpack::optimization::L_BFGS< FunctionType >::wolfe [private]

Parameter for detecting the Wolfe condition.

Definition at line 175 of file lbfgs.hpp.

Referenced by mlpack::optimization::L_BFGS< AugLagrangianFunction< mlpack::optimization::LRSDP > >::Wolfe().

21.80.4.13 template<typename FunctionType> arma::cube mlpack::optimization::L_BFGS< FunctionType >::y [private]

Stores all the y matrices in memory.

Definition at line 166 of file lbfgs.hpp.

The documentation for this class was generated from the following file:

- src/mlpack/core/optimizers/lbfgs/**lbfgs.hpp**

21.81 mlpack::optimization::LovaszThetaSDP Class Reference

This function is the Lovasz-Theta semidefinite program, as implemented in the following paper:

Public Member Functions

- **LovaszThetaSDP** ()
- **LovaszThetaSDP** (const arma::mat &**edges**)

  Initialize the Lovasz-Theta SDP with the given set of edges.
- const arma::mat & **Edges** () const
- arma::mat & **Edges** ()
- double **Evaluate** (const arma::mat &coordinates)
- double **EvaluateConstraint** (const size_t index, const arma::mat &coordinates)
- const arma::mat & **GetInitialPoint** ()
- void **Gradient** (const arma::mat &coordinates, arma::mat &gradient)
- void **GradientConstraint** (const size_t index, const arma::mat &coordinates, arma::mat &gradient)
- size_t **NumConstraints** () const

Private Attributes

- arma::mat **edges**
- arma::mat **initialPoint**
- size_t **vertices**

21.81.1 Detailed Description

This function is the Lovasz-Theta semidefinite program, as implemented in the following paper:

S. Burer, R. Monteiro "A nonlinear programming algorithm for solving semidefinite programs via low-rank factorization." Journal of Mathematical Programming, 2004

Given a simple, undirected graph G = (V, E), the Lovasz-Theta SDP is defined by:

min_X{Tr(-(e e^T)^T X) : Tr(X) = 1, X_ij = 0 for all (i, j) in E, X >= 0}

where e is the vector of all ones and X has dimension |V| x |V|.

In the Monteiro-Burer formulation, we take X = R * R^T, where R is the coordinates given to the **Evaluate()** (p. 388), **Gradient()** (p. 388), **EvaluateConstraint()** (p. 388), and **GradientConstraint()** (p. 388) functions.

Definition at line 115 of file aug_lagrangian_test_functions.hpp.

21.81.2 Constructor & Destructor Documentation

21.81.2.1 mlpack::optimization::LovaszThetaSDP::LovaszThetaSDP ( )

21.81.2.2 mlpack::optimization::LovaszThetaSDP::LovaszThetaSDP (const arma::mat & *edges*)

Initialize the Lovasz-Theta SDP with the given set of edges.

The edge matrix should consist of rows of two dimensions, where dimension 0 is the first vertex of the edge and dimension 1 is the second edge (or vice versa, as it doesn't make a difference).

Parameters

edges	Matrix of edges.

21.81.3 Member Function Documentation

21.81.3.1 const arma::mat& mlpack::optimization::LovaszThetaSDP::Edges ( ) const [inline]

Definition at line 142 of file aug_lagrangian_test_functions.hpp.

References edges.

21.81.3.2 arma::mat& mlpack::optimization::LovaszThetaSDP::Edges ( ) [inline]

Definition at line 143 of file aug_lagrangian_test_functions.hpp.

References edges.

21.81.3.3 double mlpack::optimization::LovaszThetaSDP::Evaluate (const arma::mat & *coordinates*)

21.81.3.4 double mlpack::optimization::LovaszThetaSDP::EvaluateConstraint (const size_t *index,* const arma::mat & *coordinates*)

21.81.3.5 const arma::mat& mlpack::optimization::LovaszThetaSDP::GetInitialPoint ( )

21.81.3.6 void mlpack::optimization::LovaszThetaSDP::Gradient (const arma::mat & *coordinates,* arma::mat & *gradient*)

21.81.3.7 void mlpack::optimization::LovaszThetaSDP::GradientConstraint (const size_t *index,* const arma::mat & *coordinates,* arma::mat & *gradient*)

21.81.3.8 size_t mlpack::optimization::LovaszThetaSDP::NumConstraints ( ) const

21.81.4 Member Data Documentation

21.81.4.1 arma::mat mlpack::optimization::LovaszThetaSDP::edges [private]

Definition at line 146 of file aug_lagrangian_test_functions.hpp.

Referenced by Edges().

21.81.4.2 arma::mat mlpack::optimization::LovaszThetaSDP::initialPoint [private]

Definition at line 149 of file aug_lagrangian_test_functions.hpp.

21.81.4.3 size_t mlpack::optimization::LovaszThetaSDP::vertices [private]

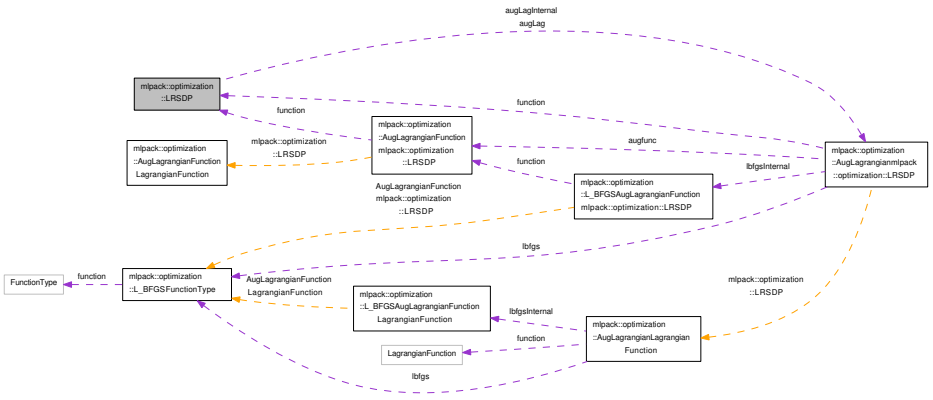
Definition at line 147 of file aug_lagrangian_test_functions.hpp.

The documentation for this class was generated from the following file:

- src/mlpack/core/optimizers/aug_lagrangian/**aug_lagrangian_test_functions.hpp**

21.82 mlpack::optimization::LRSDP Class Reference

Collaboration diagram for mlpack::optimization::LRSDP:



Public Member Functions

- **LRSDP** (const size_t numConstraints, const arma::mat &**initialPoint**)

  *Create an **LRSDP** (p. 389) to be optimized.*
- **LRSDP** (const size_t numConstraints, const arma::mat &**initialPoint**, **AugLagrangian**< **LRSDP** > &augLagrangian)

  *Create an **LRSDP** (p. 389) to be optimized, passing in an already-created **AugLagrangian** (p. 363) object.*
- const std::vector< arma::mat > & **A** () const

  Return the vector of A matrices (which correspond to the constraints).
- std::vector< arma::mat > & **A** ()

  Modify the veector of A matrices (which correspond to the constraints).
- const arma::uvec & **AModes** () const

  Return the vector of modes for the A matrices.
- arma::uvec & **AModes** ()

  Modify the vector of modes for the A matrices.
- const **AugLagrangian**< **LRSDP** > & **AugLag** () const

  Return the augmented Lagrangian object.
- **AugLagrangian**< **LRSDP** > & **AugLag** ()

  Modify the augmented Lagrangian object.
- const arma::vec & **B** () const

  Return the vector of B values.
- arma::vec & **B** ()

  Modify the vector of B values.

- const arma::mat & **C** () const

  Return the objective function matrix (C).
- arma::mat & **C** ()

  Modify the objective function matrix (C).
- double **Evaluate** (const arma::mat &coordinates) const

  *Evaluate the objective function of the **LRSDP** (p. 389) (no constraints) at the given coordinates.*
- double **EvaluateConstraint** (const size_t index, const arma::mat &coordinates) const

  *Evaluate a particular constraint of the **LRSDP** (p. 389) at the given coordinates.*
- const arma::mat & **GetInitialPoint** ()

  *Get the initial point of the **LRSDP** (p. 389).*
- void **Gradient** (const arma::mat &coordinates, arma::mat &gradient) const

  *Evaluate the gradient of the **LRSDP** (p. 389) (no constraints) at the given coordinates.*
- void **GradientConstraint** (const size_t index, const arma::mat &coordinates, arma::mat &gradient) const

  *Evaluate the gradient of a particular constraint of the **LRSDP** (p. 389) at the given coordinates.*
- size_t **NumConstraints** () const

  *Get the number of constraints in the **LRSDP** (p. 389).*
- double **Optimize** (arma::mat &coordinates)

  *Optimize the **LRSDP** (p. 389) and return the final objective value.*

Private Attributes

- std::vector< arma::mat > **a**

  A_i for each constraint.
- arma::uvec **aModes**

  1 if entries in matrix, 0 for normal.
- **AugLagrangian**< **LRSDP** > & **augLag**

  *The **AugLagrangian** (p. 363) object which will be used for optimization.*
- **AugLagrangian**< **LRSDP** > **augLagInternal**

  *Internal **AugLagrangian** (p. 363) object, if one was not passed at construction time.*
- arma::vec **b**

  b_i for each constraint.
- arma::mat **c**

  For objective function.
- arma::mat **initialPoint**

  Initial point.

21.82.1 Detailed Description

Definition at line 32 of file lrsdp.hpp.

21.82.2 Constructor & Destructor Documentation

21.82.2.1 mlpack::optimization::LRSDP::LRSDP (const size_t *numConstraints*, const arma::mat & *initialPoint*)

Create an **LRSDP** (p. 389) to be optimized.

The solution will end up being a matrix of size (rank) x (rows). To construct each constraint and the objective function, use the functions **A()** (p. 391), **B()** (p. 392), and **C()** (p. 392) to set them correctly.

Parameters

numConstraints	Number of constraints in the problem.
rank	Rank of the solution (<= rows).
rows	Number of rows in the solution.

21.82.2.2 mlpack::optimization::LRSDP::LRSDP (const size_t *numConstraints,* const arma::mat & *initialPoint,* AugLagrangian< LRSDP > & *augLagrangian*)

Create an **LRSDP** (p. 389) to be optimized, passing in an already-created **AugLagrangian** (p. 363) object.

The given initial point should be set to the size (rows) x (rank), where (rank) is the reduced rank of the problem.

Parameters

numConstraints	Number of constraints in the problem.
initialPoint	Initial point of the optimization.
auglag	Pre-initialized AugLagrangian<LRSDP> object.

21.82.3 Member Function Documentation

21.82.3.1 const std::vector<arma::mat>& mlpack::optimization::LRSDP::A ( ) const [inline]

Return the vector of A matrices (which correspond to the constraints).

Definition at line 106 of file lrsdp.hpp.

References a.

21.82.3.2 std::vector<arma::mat>& mlpack::optimization::LRSDP::A ( ) [inline]

Modify the veector of A matrices (which correspond to the constraints).

Definition at line 108 of file lrsdp.hpp.

References a.

21.82.3.3 const arma::uvec& mlpack::optimization::LRSDP::AModes ( ) const [inline]

Return the vector of modes for the A matrices.

Definition at line 111 of file lrsdp.hpp.

References aModes.

21.82.3.4 arma::uvec& mlpack::optimization::LRSDP::AModes ( ) [inline]

Modify the vector of modes for the A matrices.

Definition at line 113 of file lrsdp.hpp.

References aModes.

21.82.3.5 const AugLagrangian<LRSDP>& mlpack::optimization::LRSDP::AugLag ( ) const [inline]

Return the augmented Lagrangian object.

Definition at line 121 of file lrsdp.hpp.

References augLag.

21.82.3.6 AugLagrangian<LRSDP>& mlpack::optimization::LRSDP::AugLag ( ) [inline]

Modify the augmented Lagrangian object.

Definition at line 123 of file lrsdp.hpp.

References augLag.

21.82.3.7 const arma::vec& mlpack::optimization::LRSDP::B ( ) const [inline]

Return the vector of B values.

Definition at line 116 of file lrsdp.hpp.

References b.

21.82.3.8 arma::vec& mlpack::optimization::LRSDP::B ( ) [inline]

Modify the vector of B values.

Definition at line 118 of file lrsdp.hpp.

References b.

21.82.3.9 const arma::mat& mlpack::optimization::LRSDP::C ( ) const [inline]

Return the objective function matrix (C).

Definition at line 101 of file lrsdp.hpp.

References c.

21.82.3.10 arma::mat& mlpack::optimization::LRSDP::C ( ) [inline]

Modify the objective function matrix (C).

Definition at line 103 of file lrsdp.hpp.

References c.

21.82.3.11 double mlpack::optimization::LRSDP::Evaluate (const arma::mat & *coordinates*) const

Evaluate the objective function of the **LRSDP** (p. 389) (no constraints) at the given coordinates.

This is used by AugLagrangian<LRSDP>.

21.82.3.12 double mlpack::optimization::LRSDP::EvaluateConstraint (const size_t *index*, const arma::mat & *coordinates*) const

Evaluate a particular constraint of the **LRSDP** (p. 389) at the given coordinates.

21.82.3.13 const arma::mat& mlpack::optimization::LRSDP::GetInitialPoint ( )

Get the initial point of the **LRSDP** (p. 389).

21.82.3.14 void mlpack::optimization::LRSDP::Gradient (const arma::mat & *coordinates*, arma::mat & *gradient*) const

Evaluate the gradient of the **LRSDP** (p. 389) (no constraints) at the given coordinates.

This is used by AugLagrangian<LRSDP>.

21.82.3.15 void mlpack::optimization::LRSDP::GradientConstraint (const size_t *index*, const arma::mat & *coordinates*, arma::mat & *gradient*) const

Evaluate the gradient of a particular constraint of the **LRSDP** (p. 389) at the given coordinates.

21.82.3.16 size_t mlpack::optimization::LRSDP::NumConstraints ( ) const [inline]

Get the number of constraints in the **LRSDP** (p. 389).

Definition at line 95 of file lrsdp.hpp.

References b.

21.82.3.17 double mlpack::optimization::LRSDP::Optimize (arma::mat & *coordinates*)

Optimize the **LRSDP** (p. 389) and return the final objective value.

The given coordinates will be modified to contain the final solution.

Parameters

coordinates	Starting coordinates for the optimization.

21.82.4 Member Data Documentation

21.82.4.1 std::vector<arma::mat> mlpack::optimization::LRSDP::a [private]

A_i for each constraint.

Definition at line 131 of file lrsdp.hpp.

Referenced by A().

21.82.4.2 arma::uvec mlpack::optimization::LRSDP::aModes [private]

1 if entries in matrix, 0 for normal.

Definition at line 136 of file lrsdp.hpp.

Referenced by AModes().

21.82.4.3 AugLagrangian<LRSDP>& mlpack::optimization::LRSDP::augLag [private]

The **AugLagrangian** (p. 363) object which will be used for optimization.

Definition at line 145 of file lrsdp.hpp.

Referenced by AugLag().

21.82.4.4 AugLagrangian<LRSDP> mlpack::optimization::LRSDP::augLagInternal [private]

Internal **AugLagrangian** (p. 363) object, if one was not passed at construction time.

Definition at line 142 of file lrsdp.hpp.

21.82.4.5 arma::vec mlpack::optimization::LRSDP::b [private]

b_i for each constraint.

Definition at line 133 of file lrsdp.hpp.

Referenced by B(), and NumConstraints().

21.82.4.6 arma::mat mlpack::optimization::LRSDP::c [private]

For objective function.

Definition at line 129 of file lrsdp.hpp.

Referenced by C().

21.82.4.7 arma::mat mlpack::optimization::LRSDP::initialPoint [private]

Initial point.

Definition at line 139 of file lrsdp.hpp.

The documentation for this class was generated from the following file:

- src/mlpack/core/optimizers/lrsdp/**lrsdp.hpp**

21.83 mlpack::optimization::SGD< DecomposableFunctionType > Class Template Reference

Stochastic Gradient Descent is a technique for minimizing a function which can be expressed as a sum of other functions.

Public Member Functions

- **SGD** (DecomposableFunctionType &**function**, const double **stepSize**=0.01, const size_t **maxIterations**=100000, const double **tolerance**=1e-5, const bool shuffle=true)

  *Construct the **SGD** (p. 394) optimizer with the given function and parameters.*
- const DecomposableFunctionType & **Function** () const

  Get the instantiated function to be optimized.

- DecomposableFunctionType & **Function** ()

  Modify the instantiated function.
- size_t **MaxIterations** () const

  Get the maximum number of iterations (0 indicates no limit).
- size_t & **MaxIterations** ()

  Modify the maximum number of iterations (0 indicates no limit).
- double **Optimize** (arma::mat &iterate)

  Optimize the given function using stochastic gradient descent.
- bool **Shuffle** () const

  Get whether or not the individual functions are shuffled.
- bool & **Shuffle** ()

  Modify whether or not the individual functions are shuffled.
- double **StepSize** () const

  Get the step size.
- double & **StepSize** ()

  Modify the step size.
- double **Tolerance** () const

  Get the tolerance for termination.
- double & **Tolerance** ()

  Modify the tolerance for termination.

Private Attributes

- DecomposableFunctionType & **function**

  The instantiated function.
- size_t **maxIterations**

  The maximum number of allowed iterations.
- bool **shuffle**

  Controls whether or not the individual functions are shuffled when iterating.
- double **stepSize**

  The step size for each example.
- double **tolerance**

  The tolerance for termination.

21.83.1 Detailed Description

template<typename DecomposableFunctionType>class mlpack::optimization::SGD< DecomposableFunctionType >

Stochastic Gradient Descent is a technique for minimizing a function which can be expressed as a sum of other functions.

That is, suppose we have

$$f(A) = \sum_{i=0}^{n} f_i(A)$$

and our task is to minimize A. Stochastic gradient descent iterates over each function $f_i(A)$, producing the following update scheme:

$$A_{j+1} = A_j + \alpha \nabla f_i(A)$$

where α is a parameter which specifies the step size. i is chosen according to j (the iteration number). The **SGD** (p. 394) class supports either scanning through each of the n functions $f_i(A)$ linearly, or in a random sequence. The algorithm continues until j reaches the maximum number of iterations – or when a full sequence of updates through each of the n functions $f_i(A)$ produces an improvement within a certain tolerance ε. That is,

$$|f(A_{j+n}) - f(A_j)| < \varepsilon.$$

The parameter ε is specified by the tolerance parameter to the constructor; n is specified by the maxIterations parameter.

This class is useful for data-dependent functions whose objective function can be expressed as a sum of objective functions operating on an individual point. Then, **SGD** (p. 394) considers the gradient of the objective function operating on an individual point in its update of A.

For **SGD** (p. 394) to work, a DecomposableFunctionType template parameter is required. This class must implement the following function:

size_t NumFunctions(); double Evaluate(const arma::mat& coordinates, const size_t i); void Gradient(const arma::mat& coordinates, const size_t i, arma::mat& gradient);

NumFunctions() should return the number of functions (n), and in the other two functions, the parameter i refers to which individual function (or gradient) is being evaluated. So, for the case of a data-dependent function, such as NCA (see **mlpack::nca::NCA** (p. 296)), NumFunctions() should return the number of points in the dataset, and Evaluate(coordinates, 0) will evaluate the objective function on the first point in the dataset (presumably, the dataset is held internally in the DecomposableFunctionType).

Template Parameters

DecomposableFunctionType	Decomposable objective function type to be minimized.

Definition at line 84 of file sgd.hpp.

21.83.2 Constructor & Destructor Documentation

21.83.2.1 template<typename DecomposableFunctionType > mlpack::optimization::SGD< DecomposableFunctionType >::SGD (DecomposableFunctionType & *function*, const double *stepSize* = 0.01, const size_t *maxIterations* = 100000, const double *tolerance* = 1e-5, const bool *shuffle* = true)

Construct the **SGD** (p. 394) optimizer with the given function and parameters.

Parameters

function	Function to be optimized (minimized).
stepSize	Step size for each iteration.
maxIterations	Maximum number of iterations allowed (0 means no limit).
tolerance	Maximum absolute tolerance to terminate algorithm.
shuffle	If true, the function order is shuffled; otherwise, each function is visited in linear order.

21.83.3 Member Function Documentation

21.83.3.1 template<typename DecomposableFunctionType > const DecomposableFunctionType& mlpack::optimization::SGD< DecomposableFunctionType >::Function ( ) const [inline]

Get the instantiated function to be optimized.

Definition at line 115 of file sgd.hpp.

21.83.3.2 template<typename DecomposableFunctionType > DecomposableFunctionType& mlpack::optimization::SGD< DecomposableFunctionType >::Function ( ) [inline]

Modify the instantiated function.

Definition at line 117 of file sgd.hpp.

21.83.3.3 template<typename DecomposableFunctionType > size_t mlpack::optimization::SGD< DecomposableFunctionType >::MaxIterations ( ) const [inline]

Get the maximum number of iterations (0 indicates no limit).

Definition at line 125 of file sgd.hpp.

References mlpack::optimization::SGD< DecomposableFunctionType >::maxIterations.

21.83.3.4 template<typename DecomposableFunctionType > size_t& mlpack::optimization::SGD< DecomposableFunctionType >::MaxIterations ( ) [inline]

Modify the maximum number of iterations (0 indicates no limit).

Definition at line 127 of file sgd.hpp.

References mlpack::optimization::SGD< DecomposableFunctionType >::maxIterations.

21.83.3.5 template<typename DecomposableFunctionType > double mlpack::optimization::SGD< DecomposableFunctionType >::Optimize (arma::mat & *iterate*)

Optimize the given function using stochastic gradient descent.

The given starting point will be modified to store the finishing point of the algorithm, and the final objective value is returned.

Parameters

iterate	Starting point (will be modified).

Returns

Objective value of the final point.

21.83.3.6 template<typename DecomposableFunctionType > bool mlpack::optimization::SGD< DecomposableFunctionType >::Shuffle ( ) const [inline]

Get whether or not the individual functions are shuffled.

Definition at line 135 of file sgd.hpp.

References mlpack::optimization::SGD< DecomposableFunctionType >::shuffle.

21.83.3.7 template<typename DecomposableFunctionType > bool& mlpack::optimization::SGD< DecomposableFunctionType >::Shuffle ( ) [inline]

Modify whether or not the individual functions are shuffled.

Definition at line 137 of file sgd.hpp.

References mlpack::optimization::SGD< DecomposableFunctionType >::shuffle.

21.83.3.8 template<typename DecomposableFunctionType > double mlpack::optimization::SGD< DecomposableFunctionType >::StepSize ( ) const [inline]

Get the step size.

Definition at line 120 of file sgd.hpp.

References mlpack::optimization::SGD< DecomposableFunctionType >::stepSize.

21.83.3.9 template<typename DecomposableFunctionType > double& mlpack::optimization::SGD< DecomposableFunctionType >::StepSize ( ) [inline]

Modify the step size.

Definition at line 122 of file sgd.hpp.

References mlpack::optimization::SGD< DecomposableFunctionType >::stepSize.

21.83.3.10 template<typename DecomposableFunctionType > double mlpack::optimization::SGD< DecomposableFunctionType >::Tolerance ( ) const [inline]

Get the tolerance for termination.

Definition at line 130 of file sgd.hpp.

References mlpack::optimization::SGD< DecomposableFunctionType >::tolerance.

21.83.3.11 template<typename DecomposableFunctionType > double& mlpack::optimization::SGD< DecomposableFunctionType >::Tolerance ( ) [inline]

Modify the tolerance for termination.

Definition at line 132 of file sgd.hpp.

References mlpack::optimization::SGD< DecomposableFunctionType >::tolerance.

21.83.4 Member Data Documentation

21.83.4.1 template<typename DecomposableFunctionType > DecomposableFunctionType& mlpack::optimization::SGD< DecomposableFunctionType >::function [private]

The instantiated function.

Definition at line 141 of file sgd.hpp.

21.83.4.2 template<typename DecomposableFunctionType > size_t mlpack::optimization::SGD< DecomposableFunctionType >::maxIterations [private]

The maximum number of allowed iterations.

Definition at line 147 of file sgd.hpp.

Referenced by mlpack::optimization::SGD< DecomposableFunctionType >::MaxIterations().

21.83.4.3 template<typename DecomposableFunctionType > bool mlpack::optimization::SGD< DecomposableFunctionType >::shuffle [private]

Controls whether or not the individual functions are shuffled when iterating.

Definition at line 154 of file sgd.hpp.

Referenced by mlpack::optimization::SGD< DecomposableFunctionType >::Shuffle().

21.83.4.4 template<typename DecomposableFunctionType > double mlpack::optimization::SGD< DecomposableFunctionType >::stepSize [private]

The step size for each example.

Definition at line 144 of file sgd.hpp.

Referenced by mlpack::optimization::SGD< DecomposableFunctionType >::StepSize().

21.83.4.5 template<typename DecomposableFunctionType > double mlpack::optimization::SGD< DecomposableFunctionType >::tolerance [private]

The tolerance for termination.

Definition at line 150 of file sgd.hpp.

Referenced by mlpack::optimization::SGD< DecomposableFunctionType >::Tolerance().

The documentation for this class was generated from the following file:

- src/mlpack/core/optimizers/sgd/**sgd.hpp**

21.84 mlpack::optimization::test::GeneralizedRosenbrockFunction Class Reference

The Generalized Rosenbrock function in n dimensions, defined by f(x) = sum_i^{n - 1} (f(i)(x)) f_i(x) = 100 * (x_i^2 - x_{i + 1})^2 + (1 - x_i)^2 x_0 = [-1.2, 1, -1.2, 1, ...].

Public Member Functions

- **GeneralizedRosenbrockFunction** (int **n**)
- double **Evaluate** (const arma::mat &coordinates) const
- double **Evaluate** (const arma::mat &coordinates, const size_t i) const
- const arma::mat & **GetInitialPoint** () const
- void **Gradient** (const arma::mat &coordinates, arma::mat &gradient) const
- void **Gradient** (const arma::mat &coordinates, const size_t i, arma::mat &gradient) const
- size_t **NumFunctions** () const

Private Attributes

- arma::mat **initialPoint**
- int **n**

21.84.1 Detailed Description

The Generalized Rosenbrock function in n dimensions, defined by f(x) = sum_i^{n - 1} (f(i)(x)) f_i(x) = 100 ∗ (x_i^2 - x_{i + 1})^2 + (1 - x_i)^2 x_0 = [-1.2, 1, -1.2, 1, ...].

This should optimize to f(x) = 0, at x = [1, 1, 1, 1, ...].

This function can also be used for stochastic gradient descent (**SGD** (p. 394)) as a decomposable function (DecomposableFunctionType), so there are other overloads of **Evaluate()** (p. 400) and **Gradient()** (p. 400) implemented, as well as **NumFunctions()** (p. 400).

"An analysis of the behavior of a glass of genetic adaptive systems." K.A. De Jong. Ph.D. thesis, University of Michigan, 1975.

Definition at line 123 of file test_functions.hpp.

21.84.2 Constructor & Destructor Documentation

21.84.2.1 mlpack::optimization::test::GeneralizedRosenbrockFunction::GeneralizedRosenbrockFunction (int *n*)

21.84.3 Member Function Documentation

21.84.3.1 double mlpack::optimization::test::GeneralizedRosenbrockFunction::Evaluate (const arma::mat & *coordinates*) const

21.84.3.2 double mlpack::optimization::test::GeneralizedRosenbrockFunction::Evaluate (const arma::mat & *coordinates,* const size_t *i*) const

21.84.3.3 const arma::mat& mlpack::optimization::test::GeneralizedRosenbrockFunction::GetInitialPoint ( ) const

21.84.3.4 void mlpack::optimization::test::GeneralizedRosenbrockFunction::Gradient (const arma::mat & *coordinates,* arma::mat & *gradient*) const

21.84.3.5 void mlpack::optimization::test::GeneralizedRosenbrockFunction::Gradient (const arma::mat & *coordinates,* const size_t *i,* arma::mat & *gradient*) const

21.84.3.6 size_t mlpack::optimization::test::GeneralizedRosenbrockFunction::NumFunctions ( ) const [inline]

Definition at line 136 of file test_functions.hpp.

21.84.4 Member Data Documentation

21.84.4.1 arma::mat mlpack::optimization::test::GeneralizedRosenbrockFunction::initialPoint [private]

Definition at line 145 of file test_functions.hpp.

21.84.4.2 int mlpack::optimization::test::GeneralizedRosenbrockFunction::n [private]

Definition at line 146 of file test_functions.hpp.

The documentation for this class was generated from the following file:

- src/mlpack/core/optimizers/lbfgs/**test_functions.hpp**

21.85 mlpack::optimization::test::RosenbrockFunction Class Reference

The Rosenbrock function, defined by f(x) = f1(x) + f2(x) f1(x) = 100 (x2 - x1^2)^2 f2(x) = (1 - x1)^2 x_0 = [-1.2, 1].

Public Member Functions

- **RosenbrockFunction** ()
- double **Evaluate** (const arma::mat &coordinates)
- const arma::mat & **GetInitialPoint** () const
- void **Gradient** (const arma::mat &coordinates, arma::mat &gradient)

Private Attributes

- arma::mat **initialPoint**

21.85.1 Detailed Description

The Rosenbrock function, defined by f(x) = f1(x) + f2(x) f1(x) = 100 (x2 - x1^2)^2 f2(x) = (1 - x1)^2 x_0 = [-1.2, 1].

This should optimize to f(x) = 0, at x = [1, 1].

"An automatic method for finding the greatest or least value of a function." H.H. Rosenbrock. 1960. Comput. J. 3., 175-184.

Definition at line 63 of file test_functions.hpp.

21.85.2 Constructor & Destructor Documentation

21.85.2.1 mlpack::optimization::test::RosenbrockFunction::RosenbrockFunction ( )

21.85.3 Member Function Documentation

21.85.3.1 double mlpack::optimization::test::RosenbrockFunction::Evaluate (const arma::mat & *coordinates*)

21.85.3.2 const arma::mat& mlpack::optimization::test::RosenbrockFunction::GetInitialPoint ( ) const

21.85.3.3 void mlpack::optimization::test::RosenbrockFunction::Gradient (const arma::mat & *coordinates,* arma::mat & *gradient*)

21.85.4 Member Data Documentation

21.85.4.1 arma::mat mlpack::optimization::test::RosenbrockFunction::initialPoint [private]

Definition at line 74 of file test_functions.hpp.

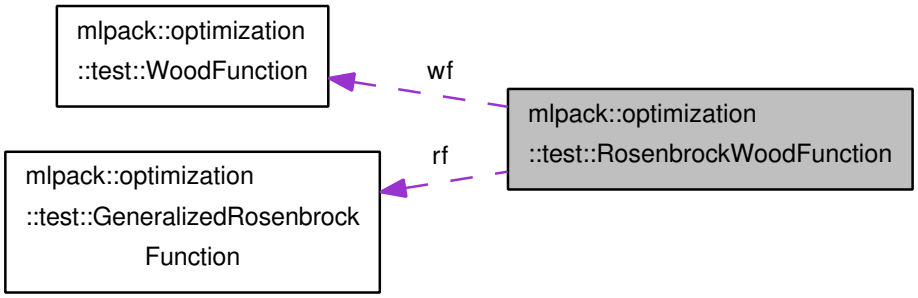
The documentation for this class was generated from the following file:

- src/mlpack/core/optimizers/lbfgs/**test_functions.hpp**

21.86 mlpack::optimization::test::RosenbrockWoodFunction Class Reference

The Generalized Rosenbrock function in 4 dimensions with the Wood Function in four dimensions.

Collaboration diagram for mlpack::optimization::test::RosenbrockWoodFunction:



Public Member Functions

- **RosenbrockWoodFunction** ()
- double **Evaluate** (const arma::mat &coordinates)
- const arma::mat & **GetInitialPoint** () const
- void **Gradient** (const arma::mat &coordinates, arma::mat &gradient)

Private Attributes

- arma::mat **initialPoint**
- **GeneralizedRosenbrockFunction rf**
- **WoodFunction wf**

21.86.1 Detailed Description

The Generalized Rosenbrock function in 4 dimensions with the Wood Function in four dimensions.

In this function we are actually optimizing a 2x4 matrix of coordinates, not a vector.

Definition at line 154 of file test_functions.hpp.

21.86.2 Constructor & Destructor Documentation

21.86.2.1 mlpack::optimization::test::RosenbrockWoodFunction::RosenbrockWoodFunction ( )

21.86.3 Member Function Documentation

21.86.3.1 double mlpack::optimization::test::RosenbrockWoodFunction::Evaluate (const arma::mat & *coordinates*)

21.86.3.2 const arma::mat& mlpack::optimization::test::RosenbrockWoodFunction::GetInitialPoint ( ) const

21.86.3.3 void mlpack::optimization::test::RosenbrockWoodFunction::Gradient (const arma::mat & *coordinates,* arma::mat & *gradient*)

21.86.4 Member Data Documentation

21.86.4.1 arma::mat mlpack::optimization::test::RosenbrockWoodFunction::initialPoint [private]

Definition at line 165 of file test_functions.hpp.

21.86.4.2 GeneralizedRosenbrockFunction mlpack::optimization::test::RosenbrockWoodFunction::rf [private]

Definition at line 166 of file test_functions.hpp.

21.86.4.3 WoodFunction mlpack::optimization::test::RosenbrockWoodFunction::wf [private]

Definition at line 167 of file test_functions.hpp.

The documentation for this class was generated from the following file:

- src/mlpack/core/optimizers/lbfgs/**test_functions.hpp**

21.87 mlpack::optimization::test::SGDTestFunction Class Reference

Very, very simple test function which is the composite of three other functions.

Public Member Functions

- **SGDTestFunction** ()

  Nothing to do for the constructor.
- double **Evaluate** (const arma::mat &coordinates, const size_t i) const

  Evaluate a function.
- arma::mat **GetInitialPoint** () const

  Get the starting point.
- void **Gradient** (const arma::mat &coordinates, const size_t i, arma::mat &gradient) const

  Evaluate the gradient of a function.
- size_t **NumFunctions** () const

  Return 3 (the number of functions).

21.87.1 Detailed Description

Very, very simple test function which is the composite of three other functions.

It turns out that although this function is very simple, optimizing it fully can take a very long time. It seems to take in excess of 10 million iterations with a step size of 0.0005.

Definition at line 35 of file test_function.hpp.

21.87.2 Constructor & Destructor Documentation

21.87.2.1 mlpack::optimization::test::SGDTestFunction::SGDTestFunction ( ) [inline]

Nothing to do for the constructor.

Definition at line 39 of file test_function.hpp.

21.87.3 Member Function Documentation

21.87.3.1 double mlpack::optimization::test::SGDTestFunction::Evaluate (const arma::mat & *coordinates,* const size_t *i*) const

Evaluate a function.

21.87.3.2 arma::mat mlpack::optimization::test::SGDTestFunction::GetInitialPoint ( ) const [inline]

Get the starting point.

Definition at line 45 of file test_function.hpp.

21.87.3.3 void mlpack::optimization::test::SGDTestFunction::Gradient (const arma::mat & *coordinates,* const size_t *i,* arma::mat & *gradient*) const

Evaluate the gradient of a function.

21.87.3.4 size_t mlpack::optimization::test::SGDTestFunction::NumFunctions ( ) const [inline]

Return 3 (the number of functions).

Definition at line 42 of file test_function.hpp.

The documentation for this class was generated from the following file:

- src/mlpack/core/optimizers/sgd/**test_function.hpp**

21.88 mlpack::optimization::test::WoodFunction Class Reference

The Wood function, defined by f(x) = f1(x) + f2(x) + f3(x) + f4(x) + f5(x) + f6(x) f1(x) = 100 (x2 - x1^2)^2 f2(x) = (1 - x1)^2 f3(x) = 90 (x4 - x3^2)^2 f4(x) = (1 - x3)^2 f5(x) = 10 (x2 + x4 - 2)^2 f6(x) = (1 / 10) (x2 - x4)^2 x_0 = [-3, -1, -3, -1].

Public Member Functions

- **WoodFunction** ()
- double **Evaluate** (const arma::mat &coordinates)
- const arma::mat & **GetInitialPoint** () const
- void **Gradient** (const arma::mat &coordinates, arma::mat &gradient)

Private Attributes

- arma::mat **initialPoint**

21.88.1 Detailed Description

The Wood function, defined by f(x) = f1(x) + f2(x) + f3(x) + f4(x) + f5(x) + f6(x) f1(x) = 100 (x2 - x1^2)^2 f2(x) = (1 - x1)^2 f3(x) = 90 (x4 - x3^2)^2 f4(x) = (1 - x3)^2 f5(x) = 10 (x2 + x4 - 2)^2 f6(x) = (1 / 10) (x2 - x4)^2 x_0 = [-3, -1, -3, -1].

This should optimize to f(x) = 0, at x = [1, 1, 1, 1].

"A comparative study of nonlinear programming codes." A.R. Colville. 1968. Rep. 320-2949, IBM N.Y. Scientific Center.

Definition at line 93 of file test_functions.hpp.

21.88.2 Constructor & Destructor Documentation

21.88.2.1 mlpack::optimization::test::WoodFunction::WoodFunction ( )

21.88.3 Member Function Documentation

21.88.3.1 double mlpack::optimization::test::WoodFunction::Evaluate (const arma::mat & *coordinates*)

21.88.3.2 const arma::mat& mlpack::optimization::test::WoodFunction::GetInitialPoint ( ) const

21.88.3.3 void mlpack::optimization::test::WoodFunction::Gradient (const arma::mat & *coordinates,* arma::mat & *gradient*)

21.88.4 Member Data Documentation

21.88.4.1 arma::mat mlpack::optimization::test::WoodFunction::initialPoint [private]

Definition at line 104 of file test_functions.hpp.

The documentation for this class was generated from the following file:

- src/mlpack/core/optimizers/lbfgs/**test_functions.hpp**

21.89 mlpack::ParamData Struct Reference

Aids in the extensibility of **CLI** (p. 132) by focusing potential changes into one structure.

Public Attributes

- std::string **desc**

  Description of this parameter, if any.
- bool **isFlag**

  True if the wasPassed value should not be ignored.
- std::string **name**

  Name of this parameter.
- std::string **tname**

  Type information of this parameter.
- boost::any **value**

  The actual value of this parameter.
- bool **wasPassed**

  True if this parameter was passed in via command line or file.

21.89.1 Detailed Description

Aids in the extensibility of **CLI** (p. 132) by focusing potential changes into one structure.

Definition at line 390 of file cli.hpp.

21.89.2 Member Data Documentation

21.89.2.1 std::string mlpack::ParamData::desc

Description of this parameter, if any.

Definition at line 395 of file cli.hpp.

21.89.2.2 bool mlpack::ParamData::isFlag

True if the wasPassed value should not be ignored.

Definition at line 403 of file cli.hpp.

21.89.2.3 std::string mlpack::ParamData::name

Name of this parameter.

Definition at line 393 of file cli.hpp.

21.89.2.4 std::string mlpack::ParamData::tname

Type information of this parameter.

Definition at line 397 of file cli.hpp.

21.89.2.5 boost::any mlpack::ParamData::value

The actual value of this parameter.

Definition at line 399 of file cli.hpp.

21.89.2.6 bool mlpack::ParamData::wasPassed

True if this parameter was passed in via command line or file.

Definition at line 401 of file cli.hpp.

The documentation for this struct was generated from the following file:

- src/mlpack/core/util/**cli.hpp**

21.90 mlpack::pca::PCA Class Reference

This class implements principal components analysis (**PCA** (p. 407)).

Public Member Functions

- **PCA** (const bool **scaleData**=false)

  *Create the **PCA** (p. 407) object, specifying if the data should be scaled in each dimension by standard deviation when **PCA** (p. 407) is performed.*
- void **Apply** (const arma::mat &data, arma::mat &transformedData, arma::vec &eigval, arma::mat &eigvec) const

  Apply Principal Component Analysis to the provided data set.
- void **Apply** (const arma::mat &data, arma::mat &transformedData, arma::vec &eigVal) const

  Apply Principal Component Analysis to the provided data set.
- double **Apply** (arma::mat &data, const size_t newDimension) const

  *Use **PCA** (p. 407) for dimensionality reduction on the given dataset.*
- double **Apply** (arma::mat &data, const int newDimension) const

  This overload is here to make sure int gets casted right to size_t.
- double **Apply** (arma::mat &data, const double varRetained) const

  *Use **PCA** (p. 407) for dimensionality reduction on the given dataset.*
- bool **ScaleData** () const

  *Get whether or not this **PCA** (p. 407) object will scale (by standard deviation) the data when **PCA** (p. 407) is performed.*
- bool & **ScaleData** ()

  *Modify whether or not this **PCA** (p. 407) object will scale (by standard deviation) the data when **PCA** (p. 407) is performed.*

Private Attributes

- bool **scaleData**

  *Whether or not the data will be scaled by standard deviation when **PCA** (p. 407) is performed.*

21.90.1 Detailed Description

This class implements principal components analysis (**PCA** (p. 407)).

This is a common, widely-used technique that is often used for either dimensionality reduction or transforming data into a better basis. Further information on **PCA** (p. 407) can be found in almost any statistics or machine learning textbook, and all over the internet.

Definition at line 38 of file pca.hpp.

21.90.2 Constructor & Destructor Documentation

21.90.2.1 mlpack::pca::PCA::PCA (const bool *scaleData* = `false`)

Create the **PCA** (p. 407) object, specifying if the data should be scaled in each dimension by standard deviation when **PCA** (p. 407) is performed.

Parameters

scaleData	Whether or not to scale the data.

21.90.3 Member Function Documentation

21.90.3.1 void mlpack::pca::PCA::Apply (const arma::mat & *data*, arma::mat & *transformedData*, arma::vec & *eigval*, arma::mat & *eigvec*) const

Apply Principal Component Analysis to the provided data set.

It is safe to pass the same matrix reference for both data and transformedData.

Parameters

data	Data matrix.
transformedData	Matrix to put results of **PCA** (p. 407) into.
eigval	Vector to put eigenvalues into.
eigvec	Matrix to put eigenvectors (loadings) into.

Referenced by Apply().

21.90.3.2 void mlpack::pca::PCA::Apply (const arma::mat & *data*, arma::mat & *transformedData*, arma::vec & *eigVal*) const

Apply Principal Component Analysis to the provided data set.

It is safe to pass the same matrix reference for both data and transformedData.

Parameters

data	Data matrix.
transformedData	Matrix to store results of **PCA** (p. 407) in.
eigval	Vector to put eigenvalues into.

21.90.3.3 double mlpack::pca::PCA::Apply (arma::mat & *data*, const size_t *newDimension*) const

Use **PCA** (p. 407) for dimensionality reduction on the given dataset.

This will save the newDimension largest principal components of the data and remove the rest. The parameter returned is the amount of variance of the data that is retained; this is a value between 0 and 1. For instance, a value of 0.9 indicates that 90% of the variance present in the data was retained.

Parameters

data	Data matrix.
newDimension	New dimension of the data.

Returns

Amount of the variance of the data retained (between 0 and 1).

21.90.3.4 double mlpack::pca::PCA::Apply (arma::mat & *data*, const int *newDimension*) const [inline]

This overload is here to make sure int gets casted right to size_t.

Definition at line 89 of file pca.hpp.

References Apply().

21.90.3.5 double mlpack::pca::PCA::Apply (arma::mat & *data,* const double *varRetained*) const

Use **PCA** (p. 407) for dimensionality reduction on the given dataset.

This will save as many dimensions as necessary to retain at least the given amount of variance (specified by parameter varRetained). The amount should be between 0 and 1; if the amount is 0, then only 1 dimension will be retained. If the amount is 1, then all dimensions will be retained.

The method returns the actual amount of variance retained, which will always be greater than or equal to the varRetained parameter.

Parameters

data	Data matrix.
varRetained	Lower bound on amount of variance to retain; should be between 0 and 1.

Returns

Actual amount of variance retained (between 0 and 1).

21.90.3.6 bool mlpack::pca::PCA::ScaleData ( ) const [inline]

Get whether or not this **PCA** (p. 407) object will scale (by standard deviation) the data when **PCA** (p. 407) is performed.

Definition at line 113 of file pca.hpp.

References scaleData.

21.90.3.7 bool& mlpack::pca::PCA::ScaleData ( ) [inline]

Modify whether or not this **PCA** (p. 407) object will scale (by standard deviation) the data when **PCA** (p. 407) is performed.

Definition at line 116 of file pca.hpp.

References scaleData.

21.90.4 Member Data Documentation

21.90.4.1 bool mlpack::pca::PCA::scaleData [private]

Whether or not the data will be scaled by standard deviation when **PCA** (p. 407) is performed.

Definition at line 121 of file pca.hpp.

Referenced by ScaleData().

The documentation for this class was generated from the following file:

- src/mlpack/methods/pca/**pca.hpp**

21.91 mlpack::radical::Radical Class Reference

An implementation of RADICAL, an algorithm for independent component analysis (ICA).

Public Member Functions

- **Radical** (const double **noiseStdDev**=0.175, const size_t **replicates**=30, const size_t **angles**=150, const size_t **sweeps**=0, const size_t **m**=0)

  Set the parameters to RADICAL.
- size_t **Angles** () const

  Get the number of angles considered during brute-force search.
- size_t & **Angles** ()

  Modify the number of angles considered during brute-force search.
- void **CopyAndPerturb** (arma::mat &xNew, const arma::mat &x) const

  *Make replicates of each data point (the number of replicates is set in either the constructor or with **Replicates()** (p. 413)) and perturb data with Gaussian noise with standard deviation noiseStdDev.*
- void **DoRadical** (const arma::mat &matX, arma::mat &matY, arma::mat &matW)

  Run RADICAL.
- double **DoRadical2D** (const arma::mat &matX)

  Two-dimensional version of RADICAL.
- double **NoiseStdDev** () const

  Get the standard deviation of the additive Gaussian noise.
- double & **NoiseStdDev** ()

  Modify the standard deviation of the additive Gaussian noise.
- size_t **Replicates** () const

  Get the number of Gaussian-perturbed replicates used per point.
- size_t & **Replicates** ()

  Modify the number of Gaussian-perturbed replicates used per point.
- size_t **Sweeps** () const

  Get the number of sweeps.
- size_t & **Sweeps** ()

  Modify the number of sweeps.
- double **Vasicek** (arma::vec &x) const

  Vasicek's m-spacing estimator of entropy, with overlap modification from (Learned-Miller and Fisher, 2003).

Private Attributes

- size_t **angles**

  Number of angles to consider in brute-force search during Radical2D.
- arma::mat **candidate**

  Internal matrix, held as member variable to prevent memory reallocations.
- size_t **m**

  Value of m to use for Vasicek's m-spacing estimator of entropy.
- double **noiseStdDev**

  Standard deviation of the Gaussian noise added to the replicates of the data points during Radical2D.
- arma::mat **perturbed**

  Internal matrix, held as member variable to prevent memory reallocations.

- size_t **replicates**

  Number of Gaussian-perturbed replicates to use (per point) in Radical2D.

- size_t **sweeps**

  Number of sweeps; each sweep calls Radical2D once for each pair of dimensions.

21.91.1 Detailed Description

An implementation of RADICAL, an algorithm for independent component analysis (ICA).

Let X be a matrix where each column is a point and each row a dimension. The goal is to find a square unmixing matrix W such that Y = W X and the rows of Y are independent components.

For more details, see the following paper:

```
@article{learned2003ica,
  title = {ICA Using Spacings Estimates of Entropy},
  author = {Learned-Miller, E.G. and Fisher III, J.W.},
  journal = {Journal of Machine Learning Research},
  volume = {4},
  pages = {1271--1295},
  year = {2003}
}
```

Definition at line 53 of file radical.hpp.

21.91.2 Constructor & Destructor Documentation

21.91.2.1 mlpack::radical::Radical::Radical (const double *noiseStdDev* = 0.175, const size_t *replicates* = 30, const size_t *angles* = 150, const size_t *sweeps* = 0, const size_t *m* = 0)

Set the parameters to RADICAL.

Parameters

noiseStdDev	Standard deviation of the Gaussian noise added to the replicates of the data points during Radical2D
replicates	Number of Gaussian-perturbed replicates to use (per point) in Radical2D
angles	Number of angles to consider in brute-force search during Radical2D
sweeps	Number of sweeps. Each sweep calls Radical2D once for each pair of dimensions
m	The variable m from Vasicek's m-spacing estimator of entropy.

21.91.3 Member Function Documentation

21.91.3.1 size_t mlpack::radical::Radical::Angles ( ) const [inline]

Get the number of angles considered during brute-force search.

Definition at line 115 of file radical.hpp.

References angles.

21.91.3.2 size_t& mlpack::radical::Radical::Angles ( ) [inline]

Modify the number of angles considered during brute-force search.

Definition at line 117 of file radical.hpp.

References angles.

21.91.3.3 void mlpack::radical::Radical::CopyAndPerturb (arma::mat & *xNew,* const arma::mat & *x*) const

Make replicates of each data point (the number of replicates is set in either the constructor or with **Replicates()** (p. 413)) and perturb data with Gaussian noise with standard deviation noiseStdDev.

21.91.3.4 void mlpack::radical::Radical::DoRadical (const arma::mat & *matX,* arma::mat & *matY,* arma::mat & *matW*)

Run RADICAL.

Parameters

matX	Input data into the algorithm - a matrix where each column is a point and each row is a dimension.
matY	Estimated independent components - a matrix where each column is a point and each row is an estimated independent component.
matW	Estimated unmixing matrix, where matY = matW $*$ matX.

21.91.3.5 double mlpack::radical::Radical::DoRadical2D (const arma::mat & *matX*)

Two-dimensional version of RADICAL.

21.91.3.6 double mlpack::radical::Radical::NoiseStdDev ( ) const [inline]

Get the standard deviation of the additive Gaussian noise.

Definition at line 105 of file radical.hpp.

References noiseStdDev.

21.91.3.7 double& mlpack::radical::Radical::NoiseStdDev ( ) [inline]

Modify the standard deviation of the additive Gaussian noise.

Definition at line 107 of file radical.hpp.

References noiseStdDev.

21.91.3.8 size_t mlpack::radical::Radical::Replicates ( ) const [inline]

Get the number of Gaussian-perturbed replicates used per point.

Definition at line 110 of file radical.hpp.

References replicates.

21.91.3.9 size_t& mlpack::radical::Radical::Replicates ( ) [inline]

Modify the number of Gaussian-perturbed replicates used per point.

Definition at line 112 of file radical.hpp.

References replicates.

21.91.3.10 size_t mlpack::radical::Radical::Sweeps ( ) const [inline]

Get the number of sweeps.

Definition at line 120 of file radical.hpp.

References sweeps.

21.91.3.11 size_t& mlpack::radical::Radical::Sweeps ( ) [inline]

Modify the number of sweeps.

Definition at line 122 of file radical.hpp.

References sweeps.

21.91.3.12 double mlpack::radical::Radical::Vasicek (arma::vec & *x*) const

Vasicek's m-spacing estimator of entropy, with overlap modification from (Learned-Miller and Fisher, 2003).

Parameters

x	Empirical sample (one-dimensional) over which to estimate entropy.

21.91.4 Member Data Documentation

21.91.4.1 size_t mlpack::radical::Radical::angles [private]

Number of angles to consider in brute-force search during Radical2D.

Definition at line 133 of file radical.hpp.

Referenced by Angles().

21.91.4.2 arma::mat mlpack::radical::Radical::candidate [private]

Internal matrix, held as member variable to prevent memory reallocations.

Definition at line 145 of file radical.hpp.

21.91.4.3 size_t mlpack::radical::Radical::m [private]

Value of m to use for Vasicek's m-spacing estimator of entropy.

Definition at line 140 of file radical.hpp.

21.91.4.4 double mlpack::radical::Radical::noiseStdDev [private]

Standard deviation of the Gaussian noise added to the replicates of the data points during Radical2D.

Definition at line 127 of file radical.hpp.

Referenced by NoiseStdDev().

21.91.4.5 arma::mat mlpack::radical::Radical::perturbed [private]

Internal matrix, held as member variable to prevent memory reallocations.

Definition at line 143 of file radical.hpp.

21.91.4.6 size_t mlpack::radical::Radical::replicates [private]

Number of Gaussian-perturbed replicates to use (per point) in Radical2D.

Definition at line 130 of file radical.hpp.

Referenced by Replicates().

21.91.4.7 size_t mlpack::radical::Radical::sweeps [private]

Number of sweeps; each sweep calls Radical2D once for each pair of dimensions.

Definition at line 137 of file radical.hpp.

Referenced by Sweeps().

The documentation for this class was generated from the following file:

- src/mlpack/methods/radical/**radical.hpp**

21.92 mlpack::range::RangeSearch< MetricType, TreeType > Class Template Reference

The **RangeSearch** (p. 415) class is a template class for performing range searches.

Public Member Functions

- **RangeSearch** (const typename TreeType::Mat &**referenceSet**, const typename TreeType::Mat &**querySet**, const bool **naive**=false, const bool **singleMode**=false, const size_t leafSize=20, const MetricType **metric**=MetricType())

  *Initialize the **RangeSearch** (p. 415) object with a different reference set and a query set.*
- **RangeSearch** (const typename TreeType::Mat &**referenceSet**, const bool **naive**=false, const bool **singleMode**=false, const size_t leafSize=20, const MetricType **metric**=MetricType())

  *Initialize the **RangeSearch** (p. 415) object with only a reference set, which will also be used as a query set.*
- **RangeSearch** (TreeType *__referenceTree__, TreeType *__queryTree__, const typename TreeType::Mat &**referenceSet**, const typename TreeType::Mat &**querySet**, const bool **singleMode**=false, const MetricType **metric**=MetricType())

  *Initialize the **RangeSearch** (p. 415) object with the given datasets and pre-constructed trees.*
- **RangeSearch** (TreeType *__referenceTree__, const typename TreeType::Mat &**referenceSet**, const bool **singleMode**=false, const MetricType **metric**=MetricType())

  *Initialize the **RangeSearch** (p. 415) object with the given reference dataset and pre-constructed tree.*
- ~**RangeSearch** ()

  *Destroy the **RangeSearch** (p. 415) object.*
- void **Search** (const **math::Range** &range, std::vector< std::vector< size_t > > &neighbors, std::vector< std::vector< double > > &distances)

  Search for all points in the given range, returning the results in the neighbors and distances objects.

Private Attributes

- bool **hasQuerySet**

  If true, a query set was passed; if false, the query set is the reference set.
- MetricType **metric**

  Instantiated distance metric.
- bool **naive**

  If true, O(n^2) naive computation is used.
- size_t **numPrunes**

  The number of pruned nodes during computation.
- std::vector< size_t > **oldFromNewQueries**

  Mappings to old query indices (used when this object builds trees).
- std::vector< size_t > **oldFromNewReferences**

  Mappings to old reference indices (used when this object builds trees).
- TreeType::Mat **queryCopy**

  Copy of query matrix; used when a tree is built internally.
- const TreeType::Mat & **querySet**

  Query set (data should be accessed using this).
- TreeType * **queryTree**

  Query tree (may be NULL).
- TreeType::Mat **referenceCopy**

  Copy of reference matrix; used when a tree is built internally.
- const TreeType::Mat & **referenceSet**

  Reference set (data should be accessed using this).
- TreeType * **referenceTree**

  Reference tree.
- bool **singleMode**

  If true, single-tree computation is used.
- bool **treeOwner**

  If true, this object is responsible for deleting the trees.

21.92.1 Detailed Description

template<typename MetricType = mlpack::metric::EuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, RangeSearchStat>>class mlpack::range::RangeSearch< MetricType, TreeType >

The **RangeSearch** (p. 415) class is a template class for performing range searches.

It is implemented in the style of a generalized tree-independent dual-tree algorithm; for more details on the actual algorithm, see the **RangeSearchRules** (p. 422) class.

Definition at line 46 of file range_search.hpp.

21.92.2 Constructor & Destructor Documentation

21.92.2.1 template<typename MetricType = mlpack::metric::EuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, RangeSearchStat>> mlpack::range::RangeSearch< MetricType, TreeType >::RangeSearch (const typename TreeType::Mat & *referenceSet,* const typename TreeType::Mat & *querySet,* const bool *naive* = false, const bool *singleMode* = false, const size_t *leafSize* = 20, const MetricType *metric* = MetricType())

Initialize the **RangeSearch** (p. 415) object with a different reference set and a query set.

Optionally, perform the computation in naive mode or single-tree mode, and set the leaf size used for tree-building. Additionally, an instantiated metric can be given, for cases where the distance metric holds data.

This method will copy the matrices to internal copies, which are rearranged during tree-building. You can avoid this extra copy by pre-constructing the trees and passing them using a different constructor.

Parameters

referenceSet	Reference dataset.
querySet	Query dataset.
naive	Whether the computation should be done in O(n^2) naive mode.
singleMode	Whether single-tree computation should be used (as opposed to dual-tree computation).
leafSize	The leaf size to be used during tree construction.
metric	Instantiated distance metric.

21.92.2.2 template<typename MetricType = mlpack::metric::EuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, RangeSearchStat>> mlpack::range::RangeSearch< MetricType, TreeType >::RangeSearch (const typename TreeType::Mat & *referenceSet,* const bool *naive* = false, const bool *singleMode* = false, const size_t *leafSize* = 20, const MetricType *metric* = MetricType())

Initialize the **RangeSearch** (p. 415) object with only a reference set, which will also be used as a query set.

Optionally, perform the computation in naive mode or single-tree mode, and set the leaf size used for tree-building. Additionally an instantiated metric can be given, for cases where the distance metric holds data.

This method will copy the reference matrix to an internal copy, which is rearranged during tree-building. You can avoid this extra copy by pre-constructing the reference tree and passing it using a different constructor.

Parameters

referenceSet	Reference dataset.
naive	Whether the computation should be done in O(n^2) naive mode.
singleMode	Whether single-tree computation should be used (as opposed to dual-tree computation).
leafSize	The leaf size to be used during tree construction.
metric	Instantiated distance metric.

21.92.2.3 template<typename MetricType = mlpack::metric::EuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, RangeSearchStat>> mlpack::range::RangeSearch< MetricType, TreeType >::RangeSearch (TreeType * *referenceTree,* TreeType * *queryTree,* const typename TreeType::Mat & *referenceSet,* const typename TreeType::Mat & *querySet,* const bool *singleMode* = false, const MetricType *metric* = MetricType())

Initialize the **RangeSearch** (p. 415) object with the given datasets and pre-constructed trees.

It is assumed that the points in referenceSet and querySet correspond to the points in referenceTree and queryTree, respectively. Optionally, choose to use single-tree mode. Naive mode is not available as an option for this constructor; instead, to run naive computation, construct a tree with all the points in one leaf (i.e. leafSize = number of points).

Additionally, an instantiated distance metric can be given, for cases where the distance metric holds data.

There is no copying of the data matrices in this constructor (because tree-building is not necessary), so this is the constructor to use when copies absolutely must be avoided.

Note

Because tree-building (at least with BinarySpaceTree) modifies the ordering of a matrix, be sure you pass the modified matrix to this object! In addition, mapping the points of the matrix back to their original indices is not done when this constructor is used.

Parameters

referenceTree	Pre-built tree for reference points.
queryTree	Pre-built tree for query points.
referenceSet	Set of reference points corresponding to referenceTree.
querySet	Set of query points corresponding to queryTree.
singleMode	Whether single-tree computation should be used (as opposed to dual-tree computation).
metric	Instantiated distance metric.

21.92.2.4 template<typename MetricType = mlpack::metric::EuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, RangeSearchStat>> mlpack::range::RangeSearch< MetricType, TreeType >::RangeSearch (TreeType * *referenceTree,* const typename TreeType::Mat & *referenceSet,* const bool *singleMode* = `false`, const MetricType *metric* = `MetricType()`)

Initialize the **RangeSearch** (p. 415) object with the given reference dataset and pre-constructed tree.

It is assumed that the points in referenceSet correspond to the points in referenceTree. Optionally, choose to use single-tree mode. Naive mode is not available as an option for this constructor; instead, to run naive computation, construct a tree with all the points in one leaf (i.e. leafSize = number of points). Additionally, an instantiated distance metric can be given, for the case where the distance metric holds data.

There is no copying of the data matrices in this constructor (because tree-building is not necessary), so this is the constructor to use when copies absolutely must be avoided.

Note

Because tree-building (at least with BinarySpaceTree) modifies the ordering of a matrix, be sure you pass the modified matrix to this object! In addition, mapping the points of the matrix back to their original indices is not done when this constructor is used.

Parameters

referenceTree	Pre-built tree for reference points.
referenceSet	Set of reference points corresponding to referenceTree.
singleMode	Whether single-tree computation should be used (as opposed to dual-tree computation).
metric	Instantiated distance metric.

21.92.2.5 template<typename MetricType = mlpack::metric::EuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, RangeSearchStat>> mlpack::range::RangeSearch< MetricType, TreeType >::~RangeSearch ( )

Destroy the **RangeSearch** (p. 415) object.

If trees were created, they will be deleted.

21.92.3 Member Function Documentation

21.92.3.1 template<typename MetricType = mlpack::metric::EuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, RangeSearchStat>> void mlpack::range::RangeSearch< MetricType, TreeType >::Search (const math::Range & *range,* std::vector< std::vector< size_t > > & *neighbors,* std::vector< std::vector< double > > & *distances*)

Search for all points in the given range, returning the results in the neighbors and distances objects.

Each entry in the external vector corresponds to a query point. Each of these entries holds a vector which contains the indices and distances of the reference points falling into the given range.

That is:

- neighbors.size() and distances.size() both equal the number of query points.
- neighbors[i] contains the indices of all the points in the reference set which have distances inside the given range to query point i.
- distances[i] contains all of the distances corresponding to the indices contained in neighbors[i].
- neighbors[i] and distances[i] are not sorted in any particular order.

Parameters

range	Range of distances in which to search.
neighbors	Object which will hold the list of neighbors for each point which fell into the given range, for each query point.
distances	Object which will hold the list of distances for each point which fell into the given range, for each query point.

21.92.4 Member Data Documentation

21.92.4.1 template<typename MetricType = mlpack::metric::EuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, RangeSearchStat>> bool mlpack::range::RangeSearch< MetricType, TreeType >::hasQuerySet [private]

If true, a query set was passed; if false, the query set is the reference set.

Definition at line 229 of file range_search.hpp.

21.92.4.2 template<typename MetricType = mlpack::metric::EuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, RangeSearchStat>> MetricType mlpack::range::RangeSearch< MetricType, TreeType >::metric [private]

Instantiated distance metric.

Definition at line 237 of file range_search.hpp.

21.92.4.3 template<typename MetricType = mlpack::metric::EuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, RangeSearchStat>> bool mlpack::range::RangeSearch< MetricType, TreeType >::naive [private]

If true, O(n^2) naive computation is used.

Definition at line 232 of file range_search.hpp.

21.92.4.4 template<typename MetricType = mlpack::metric::EuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, RangeSearchStat>> size_t mlpack::range::RangeSearch< MetricType, TreeType >::numPrunes [private]

The number of pruned nodes during computation.

Definition at line 240 of file range_search.hpp.

21.92.4.5 template<typename MetricType = mlpack::metric::EuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, RangeSearchStat>> std::vector<size_t> mlpack::range::RangeSearch< MetricType, TreeType >::oldFromNewQueries [private]

Mappings to old query indices (used when this object builds trees).

Definition at line 223 of file range_search.hpp.

21.92.4.6 template<typename MetricType = mlpack::metric::EuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, RangeSearchStat>> std::vector<size_t> mlpack::range::RangeSearch< MetricType, TreeType >::oldFromNewReferences [private]

Mappings to old reference indices (used when this object builds trees).

Definition at line 221 of file range_search.hpp.

21.92.4.7 template<typename MetricType = mlpack::metric::EuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, RangeSearchStat>> TreeType::Mat mlpack::range::RangeSearch< MetricType, TreeType >::queryCopy [private]

Copy of query matrix; used when a tree is built internally.

Definition at line 208 of file range_search.hpp.

21.92.4.8 template<typename MetricType = mlpack::metric::EuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, RangeSearchStat>> const TreeType::Mat& mlpack::range::RangeSearch< MetricType, TreeType >::querySet [private]

Query set (data should be accessed using this).

Definition at line 213 of file range_search.hpp.

21.92.4.9 template<typename MetricType = mlpack::metric::EuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, RangeSearchStat>> TreeType* mlpack::range::RangeSearch< MetricType, TreeType >::queryTree [private]

Query tree (may be NULL).

Definition at line 218 of file range_search.hpp.

21.92.4.10 template<typename MetricType = mlpack::metric::EuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, RangeSearchStat>> TreeType::Mat mlpack::range::RangeSearch< MetricType, TreeType >::referenceCopy [private]

Copy of reference matrix; used when a tree is built internally.

Definition at line 206 of file range_search.hpp.

21.92.4.11 template<typename MetricType = mlpack::metric::EuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, RangeSearchStat>> const TreeType::Mat& mlpack::range::RangeSearch< MetricType, TreeType >::referenceSet [private]

Reference set (data should be accessed using this).

Definition at line 211 of file range_search.hpp.

21.92.4.12 template<typename MetricType = mlpack::metric::EuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, RangeSearchStat>> TreeType* mlpack::range::RangeSearch< MetricType, TreeType >::referenceTree [private]

Reference tree.

Definition at line 216 of file range_search.hpp.

21.92.4.13 template<typename MetricType = mlpack::metric::EuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, RangeSearchStat>> bool mlpack::range::RangeSearch< MetricType, TreeType >::singleMode [private]

If true, single-tree computation is used.

Definition at line 234 of file range_search.hpp.

21.92.4.14 template<typename MetricType = mlpack::metric::EuclideanDistance, typename TreeType = tree::BinarySpaceTree<bound::HRectBound<2>, RangeSearchStat>> bool mlpack::range::RangeSearch< MetricType, TreeType >::treeOwner [private]

If true, this object is responsible for deleting the trees.

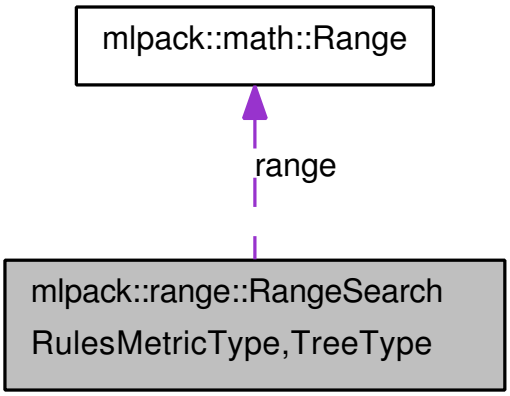
Definition at line 226 of file range_search.hpp.

The documentation for this class was generated from the following file:

- src/mlpack/methods/range_search/**range_search.hpp**

21.93 mlpack::range::RangeSearchRules< MetricType, TreeType > Class Template Reference

Collaboration diagram for mlpack::range::RangeSearchRules< MetricType, TreeType >:



Public Member Functions

- **RangeSearchRules** (const arma::mat &**referenceSet**, const arma::mat &**querySet**, const **math::Range** &**range**, std::vector< std::vector< size_t > > &**neighbors**, std::vector< std::vector< double > > &**distances**, MetricType &**metric**)

  *Construct the **RangeSearchRules** (p. 422) object.*
- double **BaseCase** (const size_t queryIndex, const size_t referenceIndex)

  Compute the base case between the given query point and reference point.
- double **Rescore** (const size_t queryIndex, TreeType &referenceNode, const double oldScore) const

  Re-evaluate the score for recursion order.
- double **Rescore** (TreeType &queryNode, TreeType &referenceNode, const double oldScore) const

  Re-evaluate the score for recursion order.
- double **Score** (const size_t queryIndex, TreeType &referenceNode)

  Get the score for recursion order.
- double **Score** (TreeType &queryNode, TreeType &referenceNode)

  Get the score for recursion order.

Private Member Functions

- void **AddResult** (const size_t queryIndex, TreeType &referenceNode)

  Add all the points in the given node to the results for the given query point.

Private Attributes

- std::vector< std::vector < double > > & **distances**

  The vector the resultant neighbor distances should be stored in.
- size_t **lastQueryIndex**

The last query index.

- size_t **lastReferenceIndex**

  The last reference index.

- MetricType & **metric**

  The instantiated metric.

- std::vector< std::vector < size_t > > & **neighbors**

  The vector the resultant neighbor indices should be stored in.

- const arma::mat & **querySet**

  The query set.

- const **math::Range** & **range**

  The range of distances for which we are searching.

- const arma::mat & **referenceSet**

  The reference set.

21.93.1 Detailed Description

template<typename MetricType, typename TreeType>class mlpack::range::RangeSearchRules< MetricType, TreeType >

Definition at line 30 of file range_search_rules.hpp.

21.93.2 Constructor & Destructor Documentation

21.93.2.1 template<typename MetricType , typename TreeType > mlpack::range::RangeSearchRules< MetricType, TreeType >::RangeSearchRules (const arma::mat & *referenceSet,* const arma::mat & *querySet,* const math::Range & *range,* std::vector< std::vector< size_t > > & *neighbors,* std::vector< std::vector< double > > & *distances,* MetricType & *metric*)

Construct the **RangeSearchRules** (p. 422) object.

This is usually done from within the **RangeSearch** (p. 415) class at search time.

Parameters

referenceSet	Set of reference data.
querySet	Set of query data.
range	Range to search for.
neighbors	Vector to store resulting neighbors in.
distances	Vector to store resulting distances in.
metric	Instantiated metric.

21.93.3 Member Function Documentation

21.93.3.1 template<typename MetricType , typename TreeType > void mlpack::range::RangeSearchRules< MetricType, TreeType >::AddResult (const size_t *queryIndex,* TreeType & *referenceNode*) [private]

Add all the points in the given node to the results for the given query point.

If the base case has already been calculated, we make sure to not add that to the results twice.

21.93.3.2 template<typename MetricType , typename TreeType > double mlpack::range::RangeSearchRules< MetricType, TreeType >::BaseCase (const size_t *queryIndex,* const size_t *referenceIndex*)

Compute the base case between the given query point and reference point.

Parameters

queryIndex	Index of query point.
referenceIndex	Index of reference point.

21.93.3.3 template<typename MetricType , typename TreeType > double mlpack::range::RangeSearchRules< MetricType, TreeType >::Rescore (const size_t *queryIndex,* TreeType & *referenceNode,* const double *oldScore*) const

Re-evaluate the score for recursion order.

A low score indicates priority for recursion, while DBL_MAX indicates that the node should not be recursed into at all (it should be pruned). This is used when the score has already been calculated, but another recursion may have modified the bounds for pruning. So the old score is checked against the new pruning bound.

Parameters

queryIndex	Index of query point.
referenceNode	Candidate node to be recursed into.
oldScore	Old score produced by **Score()** (p. 425) (or **Rescore()** (p. 425)).

21.93.3.4 template<typename MetricType , typename TreeType > double mlpack::range::RangeSearchRules< MetricType, TreeType >::Rescore (TreeType & *queryNode,* TreeType & *referenceNode,* const double *oldScore*) const

Re-evaluate the score for recursion order.

A low score indicates priority for recursion, while DBL_MAX indicates that the node should not be recursed into at all (it should be pruned). This is used when the score has already been calculated, but another recursion may have modified the bounds for pruning. So the old score is checked against the new pruning bound.

Parameters

queryNode	Candidate query node to recurse into.
referenceNode	Candidate reference node to recurse into.
oldScore	Old score produced by **Score()** (p. 425) (or **Rescore()** (p. 425)).

21.93.3.5 template<typename MetricType , typename TreeType > double mlpack::range::RangeSearchRules< MetricType, TreeType >::Score (const size_t *queryIndex,* TreeType & *referenceNode*)

Get the score for recursion order.

A low score indicates priority for recursion, while DBL_MAX indicates that the node should not be recursed into at all (it should be pruned).

Parameters

queryIndex	Index of query point.
referenceNode	Candidate node to be recursed into.

21.93.3.6 template<typename MetricType , typename TreeType > double mlpack::range::RangeSearchRules< MetricType, TreeType >::Score (TreeType & *queryNode,* TreeType & *referenceNode*)

Get the score for recursion order.

A low score indicates priority for recursion, while DBL_MAX indicates that the node should not be recursed into at all (it should be pruned).

Parameters

queryNode	Candidate query node to recurse into.
referenceNode	Candidate reference node to recurse into.

21.93.4 Member Data Documentation

21.93.4.1 template<typename MetricType , typename TreeType > std::vector<std::vector<double> >& mlpack::range::RangeSearchRules< MetricType, TreeType >::distances [private]

The vector the resultant neighbor distances should be stored in.

Definition at line 123 of file range_search_rules.hpp.

21.93.4.2 template<typename MetricType , typename TreeType > size_t mlpack::range::RangeSearchRules< MetricType, TreeType >::lastQueryIndex [private]

The last query index.

Definition at line 129 of file range_search_rules.hpp.

21.93.4.3 template<typename MetricType , typename TreeType > size_t mlpack::range::RangeSearchRules< MetricType, TreeType >::lastReferenceIndex [private]

The last reference index.

Definition at line 131 of file range_search_rules.hpp.

21.93.4.4 template<typename MetricType , typename TreeType > MetricType& mlpack::range::RangeSearchRules< MetricType, TreeType >::metric [private]

The instantiated metric.

Definition at line 126 of file range_search_rules.hpp.

21.93.4.5 template<typename MetricType , typename TreeType > std::vector<std::vector<size_t> >& mlpack::range::RangeSearchRules< MetricType, TreeType >::neighbors [private]

The vector the resultant neighbor indices should be stored in.

Definition at line 120 of file range_search_rules.hpp.

21.93.4.6 template<typename MetricType , typename TreeType > const arma::mat& mlpack::range::RangeSearchRules< MetricType, TreeType >::querySet [private]

The query set.

Definition at line 114 of file range_search_rules.hpp.

21.93.4.7 template<typename MetricType , typename TreeType > const math::Range& mlpack::range::RangeSearchRules< MetricType, TreeType >::range [private]

The range of distances for which we are searching.

Definition at line 117 of file range_search_rules.hpp.

21.93.4.8 template<typename MetricType , typename TreeType > const arma::mat& mlpack::range::RangeSearchRules< MetricType, TreeType >::referenceSet [private]

The reference set.

Definition at line 111 of file range_search_rules.hpp.

The documentation for this class was generated from the following file:

- src/mlpack/methods/range_search/**range_search_rules.hpp**

21.94 mlpack::range::RangeSearchStat Class Reference

Statistic class for **RangeSearch** (p. 415), to be set to the StatisticType of the tree type that range search is being performed with.

Public Member Functions

- **RangeSearchStat** ()

  Initialize the statistic.

- template<typename TreeType >
  RangeSearchStat (TreeType &)

  Initialize the statistic given a tree node that this statistic belongs to.

- double **LastDistance** () const

  Get the last distance evaluation.

- double & **LastDistance** ()

  Modify the last distance evaluation.

- void * **LastDistanceNode** () const

  Get the last distance evaluation node.

- void *& **LastDistanceNode** ()

  Modify the last distance evaluation node.

Private Attributes

- double **lastDistance**

  The last distance evaluation.

- void * **lastDistanceNode**

  The last distance evaluation node.

21.94.1 Detailed Description

Statistic class for **RangeSearch** (p. 415), to be set to the StatisticType of the tree type that range search is being performed with.

This class just holds the last visited node and the corresponding base case result.

Definition at line 36 of file range_search_stat.hpp.

21.94.2 Constructor & Destructor Documentation

21.94.2.1 mlpack::range::RangeSearchStat::RangeSearchStat ( ) [inline]

Initialize the statistic.

Definition at line 42 of file range_search_stat.hpp.

21.94.2.2 template<typename TreeType > mlpack::range::RangeSearchStat::RangeSearchStat (TreeType &) [inline]

Initialize the statistic given a tree node that this statistic belongs to.

In this case, we ignore the node.

Definition at line 49 of file range_search_stat.hpp.

21.94.3 Member Function Documentation

21.94.3.1 double mlpack::range::RangeSearchStat::LastDistance ( ) const [inline]

Get the last distance evaluation.

Definition at line 58 of file range_search_stat.hpp.

References lastDistance.

21.94.3.2 double& mlpack::range::RangeSearchStat::LastDistance ( ) [inline]

Modify the last distance evaluation.

Definition at line 60 of file range_search_stat.hpp.

References lastDistance.

21.94.3.3 void* mlpack::range::RangeSearchStat::LastDistanceNode ( ) const [inline]

Get the last distance evaluation node.

Definition at line 54 of file range_search_stat.hpp.

References lastDistanceNode.

21.94.3.4 void*& mlpack::range::RangeSearchStat::LastDistanceNode ( ) [inline]

Modify the last distance evaluation node.

Definition at line 56 of file range_search_stat.hpp.

References lastDistanceNode.

21.94.4 Member Data Documentation

21.94.4.1 double mlpack::range::RangeSearchStat::lastDistance [private]

The last distance evaluation.

Definition at line 66 of file range_search_stat.hpp.

Referenced by LastDistance().

21.94.4.2 void* mlpack::range::RangeSearchStat::lastDistanceNode [private]

The last distance evaluation node.

Definition at line 64 of file range_search_stat.hpp.

Referenced by LastDistanceNode().

The documentation for this class was generated from the following file:

- src/mlpack/methods/range_search/**range_search_stat.hpp**

21.95 mlpack::regression::LARS Class Reference

An implementation of **LARS** (p. 429), a stage-wise homotopy-based algorithm for l1-regularized linear regression (LASSO) and l1+l2 regularized linear regression (Elastic Net).

Public Member Functions

- **LARS** (const bool **useCholesky**, const double **lambda1**=0.0, const double **lambda2**=0.0, const double **tolerance**=1e-16)

  *Set the parameters to **LARS** (p. 429).*
- **LARS** (const bool **useCholesky**, const arma::mat &gramMatrix, const double **lambda1**=0.0, const double **lambda2**=0.0, const double **tolerance**=1e-16)

  *Set the parameters to **LARS** (p. 429), and pass in a precalculated Gram matrix.*
- const std::vector< size_t > & **ActiveSet** () const

  Access the set of active dimensions.
- const std::vector< arma::vec > & **BetaPath** () const

  Access the set of coefficients after each iteration; the solution is the last element.
- const std::vector< double > & **LambdaPath** () const

  Access the set of values for lambda1 after each iteration; the solution is the last element.
- const arma::mat & **MatUtriCholFactor** () const

  Access the upper triangular cholesky factor.
- void **Regress** (const arma::mat &data, const arma::vec &responses, arma::vec &beta, const bool transposeData=true)

  *Run **LARS** (p. 429).*

Private Member Functions

- void **Activate** (const size_t varInd)

  Add dimension varInd to active set.
- void **CholeskyDelete** (const size_t colToKill)
- void **CholeskyInsert** (const arma::vec &newX, const arma::mat &X)
- void **CholeskyInsert** (double sqNormNewX, const arma::vec &newGramCol)
- void **ComputeYHatDirection** (const arma::mat &matX, const arma::vec &betaDirection, arma::vec &yHatDirection)
- void **Deactivate** (const size_t activeVarInd)

  Remove activeVarInd'th element from active set.
- void **GivensRotate** (const arma::vec::fixed< 2 > &x, arma::vec::fixed< 2 > &rotatedX, arma::mat &G)
- void **InterpolateBeta** ()

Private Attributes

- std::vector< size_t > **activeSet**

  Active set of dimensions.
- std::vector< arma::vec > **betaPath**

  Solution path.
- bool **elasticNet**

  True if this is the elastic net problem.
- std::vector< bool > **isActive**

  Active set membership indicator (for each dimension).
- double **lambda1**

  Regularization parameter for l1 penalty.
- double **lambda2**

  Regularization parameter for l2 penalty.
- std::vector< double > **lambdaPath**

  Value of lambda_1 for each solution in solution path.
- bool **lasso**

  True if this is the LASSO problem.
- const arma::mat & **matGram**

  Reference to the Gram matrix we will use.
- arma::mat **matGramInternal**

  Gram matrix.
- arma::mat **matUtriCholFactor**

  Upper triangular cholesky factor; initially 0x0 matrix.
- double **tolerance**

  Tolerance for main loop.
- bool **useCholesky**

  Whether or not to use Cholesky decomposition when solving linear system.

21.95.1 Detailed Description

An implementation of **LARS** (p. 429), a stage-wise homotopy-based algorithm for l1-regularized linear regression (LASSO) and l1+l2 regularized linear regression (Elastic Net).

Let X be a matrix where each row is a point and each column is a dimension and let y be a vector of responses.

The Elastic Net problem is to solve

$$\min_{\beta} 0.5||X\beta - y||_2^2 + \lambda_1||\beta||_1 + 0.5\lambda_2||\beta||_2^2$$

where β is the vector of regression coefficients.

If $\lambda_1 > 0$ and $\lambda_2 = 0$, the problem is the LASSO. If $\lambda_1 > 0$ and $\lambda_2 > 0$, the problem is the elastic net. If $\lambda_1 = 0$ and $\lambda_2 > 0$, the problem is ridge regression. If $\lambda_1 = 0$ and $\lambda_2 = 0$, the problem is unregularized linear regression.

Note: This algorithm is not recommended for use (in terms of efficiency) when λ_1 = 0.

For more details, see the following papers:

```
@article{efron2004least,
  title={Least angle regression},
  author={Efron, B. and Hastie, T. and Johnstone, I. and Tibshirani, R.},
  journal={The Annals of statistics},
  volume={32},
  number={2},
  pages={407--499},
  year={2004},
  publisher={Institute of Mathematical Statistics}
}


@article{zou2005regularization,
  title={Regularization and variable selection via the elastic net},
  author={Zou, H. and Hastie, T.},
  journal={Journal of the Royal Statistical Society Series B},
  volume={67},
  number={2},
  pages={301--320},
  year={2005},
  publisher={Royal Statistical Society}
}
```

Definition at line 100 of file lars.hpp.

21.95.2 Constructor & Destructor Documentation

21.95.2.1 mlpack::regression::LARS::LARS (const bool *useCholesky*, const double *lambda1* = 0.0, const double *lambda2* = 0.0, const double *tolerance* = 1e-16)

Set the parameters to **LARS** (p. 429).

Both lambda1 and lambda2 default to 0.

Parameters

useCholesky	Whether or not to use Cholesky decomposition when solving linear system (as opposed to using the full Gram matrix).

lambda1	Regularization parameter for l1-norm penalty.
lambda2	Regularization parameter for l2-norm penalty.
tolerance	Run until the maximum correlation of elements in (X^T y) is less than this.

21.95.2.2 mlpack::regression::LARS::LARS (const bool *useCholesky,* const arma::mat & *gramMatrix,* const double *lambda1* = 0.0, const double *lambda2* = 0.0, const double *tolerance* = 1e-16)

Set the parameters to **LARS** (p. 429), and pass in a precalculated Gram matrix.

Both lambda1 and lambda2 default to 0.

Parameters

useCholesky	Whether or not to use Cholesky decomposition when solving linear system (as opposed to using the full Gram matrix).
gramMatrix	Gram matrix.
lambda1	Regularization parameter for l1-norm penalty.
lambda2	Regularization parameter for l2-norm penalty.
tolerance	Run until the maximum correlation of elements in (X^T y) is less than this.

21.95.3 Member Function Documentation

21.95.3.1 void mlpack::regression::LARS::Activate (const size_t *varInd*) [private]

Add dimension varInd to active set.

Parameters

varInd	Dimension to add to active set.

21.95.3.2 const std::vector<size_t>& mlpack::regression::LARS::ActiveSet ( ) const [inline]

Access the set of active dimensions.

Definition at line 156 of file lars.hpp.

References activeSet.

21.95.3.3 const std::vector<arma::vec>& mlpack::regression::LARS::BetaPath ( ) const [inline]

Access the set of coefficients after each iteration; the solution is the last element.

Definition at line 160 of file lars.hpp.

References betaPath.

21.95.3.4 void mlpack::regression::LARS::CholeskyDelete (const size_t *colToKill*) [private]

21.95.3.5 void mlpack::regression::LARS::CholeskyInsert (const arma::vec & *newX,* const arma::mat & *X*) [private]

21.95.3.6 void mlpack::regression::LARS::CholeskyInsert (double *sqNormNewX,* const arma::vec & *newGramCol*) [private]

21.95.3.7 void mlpack::regression::LARS::ComputeYHatDirection (const arma::mat & *matX*, const arma::vec & *betaDirection*, arma::vec & *yHatDirection*) [private]

21.95.3.8 void mlpack::regression::LARS::Deactivate (const size_t *activeVarInd*) [private]

Remove activeVarInd'th element from active set.

Parameters

activeVarInd	Index of element to remove from active set.

21.95.3.9 void mlpack::regression::LARS::GivensRotate (const arma::vec::fixed< 2 > & *x*, arma::vec::fixed< 2 > & *rotatedX*, arma::mat & *G*) [private]

21.95.3.10 void mlpack::regression::LARS::InterpolateBeta ( ) [private]

21.95.3.11 const std::vector<double>& mlpack::regression::LARS::LambdaPath ( ) const [inline]

Access the set of values for lambda1 after each iteration; the solution is the last element.

Definition at line 164 of file lars.hpp.

References lambdaPath.

21.95.3.12 const arma::mat& mlpack::regression::LARS::MatUtriCholFactor ( ) const [inline]

Access the upper triangular cholesky factor.

Definition at line 167 of file lars.hpp.

References matUtriCholFactor.

21.95.3.13 void mlpack::regression::LARS::Regress (const arma::mat & *data*, const arma::vec & *responses*, arma::vec & *beta*, const bool *transposeData* = true)

Run **LARS** (p. 429).

The input matrix (like all MLPACK matrices) should be column-major – each column is an observation and each row is a dimension. However, because **LARS** (p. 429) is more efficient on a row-major matrix, this method will (internally) transpose the matrix. If this transposition is not necessary (i.e., you want to pass in a row-major matrix), pass 'false' for the transposeData parameter.

Parameters

data	Column-major input data (or row-major input data if rowMajor = true).
responses	A vector of targets.
beta	Vector to store the solution (the coefficients) in.
rowMajor	Set to false if the data is row-major.

21.95.4 Member Data Documentation

21.95.4.1 std::vector<size_t> mlpack::regression::LARS::activeSet [private]

Active set of dimensions.

Definition at line 202 of file lars.hpp.

Referenced by ActiveSet().

21.95.4.2 std::vector<arma::vec> mlpack::regression::LARS::betaPath [private]

Solution path.

Definition at line 196 of file lars.hpp.

Referenced by BetaPath().

21.95.4.3 bool mlpack::regression::LARS::elasticNet [private]

True if this is the elastic net problem.

Definition at line 188 of file lars.hpp.

21.95.4.4 std::vector<bool> mlpack::regression::LARS::isActive [private]

Active set membership indicator (for each dimension).

Definition at line 205 of file lars.hpp.

21.95.4.5 double mlpack::regression::LARS::lambda1 [private]

Regularization parameter for l1 penalty.

Definition at line 185 of file lars.hpp.

21.95.4.6 double mlpack::regression::LARS::lambda2 [private]

Regularization parameter for l2 penalty.

Definition at line 190 of file lars.hpp.

21.95.4.7 std::vector<double> mlpack::regression::LARS::lambdaPath [private]

Value of lambda_1 for each solution in solution path.

Definition at line 199 of file lars.hpp.

Referenced by LambdaPath().

21.95.4.8 bool mlpack::regression::LARS::lasso [private]

True if this is the LASSO problem.

Definition at line 183 of file lars.hpp.

21.95.4.9 const arma::mat& mlpack::regression::LARS::matGram [private]

Reference to the Gram matrix we will use.

Definition at line 174 of file lars.hpp.

21.95.4.10 arma::mat mlpack::regression::LARS::matGramInternal [private]

Gram matrix.

Definition at line 171 of file lars.hpp.

21.95.4.11 arma::mat mlpack::regression::LARS::matUtriCholFactor [private]

Upper triangular cholesky factor; initially 0x0 matrix.

Definition at line 177 of file lars.hpp.

Referenced by MatUtriCholFactor().

21.95.4.12 double mlpack::regression::LARS::tolerance [private]

Tolerance for main loop.

Definition at line 193 of file lars.hpp.

21.95.4.13 bool mlpack::regression::LARS::useCholesky [private]

Whether or not to use Cholesky decomposition when solving linear system.

Definition at line 180 of file lars.hpp.

The documentation for this class was generated from the following file:

- src/mlpack/methods/lars/**lars.hpp**

21.96 mlpack::regression::LinearRegression Class Reference

A simple linear regression algorithm using ordinary least squares.

Public Member Functions

- **LinearRegression** (const arma::mat &predictors, const arma::vec &responses, const double **lambda**=0)
  Creates the model.
- **LinearRegression** (const std::string &filename)
  Initialize the model from a file.
- **LinearRegression** (const **LinearRegression** &linearRegression)
  Copy constructor.
- **LinearRegression** ()
  Empty constructor.
- double **ComputeError** (const arma::mat &points, const arma::vec &responses) const
  Calculate the L2 squared error on the given predictors and responses using this linear regression model.
- double **Lambda** () const
  Return the Tikhonov regularization parameter for ridge regression.

- double & **Lambda** ()

  Modify the Tikhonov regularization parameter for ridge regression.
- const arma::vec & **Parameters** () const

  Return the parameters (the b vector).
- arma::vec & **Parameters** ()

  Modify the parameters (the b vector).
- void **Predict** (const arma::mat &points, arma::vec &predictions) const

  Calculate y_i for each data point in points.

Private Attributes

- double **lambda**

  The Tikhonov regularization parameter for ridge regression (0 for linear regression).
- arma::vec **parameters**

  The calculated B.

21.96.1 Detailed Description

A simple linear regression algorithm using ordinary least squares.

Optionally, this class can perform ridge regression, if the lambda parameter is set to a number greater than zero.

Definition at line 35 of file linear_regression.hpp.

21.96.2 Constructor & Destructor Documentation

21.96.2.1 mlpack::regression::LinearRegression::LinearRegression (const arma::mat & *predictors,* const arma::vec & *responses,* const double *lambda* = 0)

Creates the model.

Parameters

predictors	X, matrix of data points to create B with.
responses	y, the measured data for each point in X

21.96.2.2 mlpack::regression::LinearRegression::LinearRegression (const std::string & *filename*)

Initialize the model from a file.

Parameters

filename	the name of the file to load the model from.

21.96.2.3 mlpack::regression::LinearRegression::LinearRegression (const LinearRegression & *linearRegression*)

Copy constructor.

Parameters

linearRegression	the other instance to copy parameters from.

21.96.2.4 mlpack::regression::LinearRegression::LinearRegression ( ) [inline]

Empty constructor.

Definition at line 65 of file linear_regression.hpp.

21.96.3 Member Function Documentation

21.96.3.1 double mlpack::regression::LinearRegression::ComputeError (const arma::mat & *points,* const arma::vec & *responses*) const

Calculate the L2 squared error on the given predictors and responses using this linear regression model.

This calculation returns

$$(1/n) * \|y - XB\|_2^2$$

where y is the responses vector, X is the matrix of predictors, and B is the parameters of the trained linear regression model.

As this number decreases to 0, the linear regression fit is better.

Parameters

predictors	Matrix of predictors (X).
responses	Vector of responses (y).

21.96.3.2 double mlpack::regression::LinearRegression::Lambda ( ) const [inline]

Return the Tikhonov regularization parameter for ridge regression.

Definition at line 101 of file linear_regression.hpp.

References lambda.

21.96.3.3 double& mlpack::regression::LinearRegression::Lambda ( ) [inline]

Modify the Tikhonov regularization parameter for ridge regression.

Definition at line 103 of file linear_regression.hpp.

References lambda.

21.96.3.4 const arma::vec& mlpack::regression::LinearRegression::Parameters ( ) const [inline]

Return the parameters (the b vector).

Definition at line 96 of file linear_regression.hpp.

References parameters.

21.96.3.5 arma::vec& mlpack::regression::LinearRegression::Parameters ( ) [inline]

Modify the parameters (the b vector).

Definition at line 98 of file linear_regression.hpp.

References parameters.

21.96.3.6 void mlpack::regression::LinearRegression::Predict (const arma::mat & *points,* arma::vec & *predictions*) const

Calculate y_i for each data point in points.

Parameters

points	the data points to calculate with.
predictions	y, will contain calculated values on completion.

21.96.4 Member Data Documentation

21.96.4.1 double mlpack::regression::LinearRegression::lambda [private]

The Tikhonov regularization parameter for ridge regression (0 for linear regression).

Definition at line 116 of file linear_regression.hpp.

Referenced by Lambda().

21.96.4.2 arma::vec mlpack::regression::LinearRegression::parameters [private]

The calculated B.

Initialized and filled by constructor to hold the least squares solution.

Definition at line 110 of file linear_regression.hpp.

Referenced by Parameters().

The documentation for this class was generated from the following file:

- src/mlpack/methods/linear_regression/**linear_regression.hpp**

21.97 mlpack::regression::LogisticRegression< OptimizerType > Class Template Reference

Public Member Functions

- **LogisticRegression** (const arma::mat &predictors, const arma::vec &responses, const double **lambda**=0)

  *Construct the **LogisticRegression*** (p. *438*) *class with the given labeled training data.*
- **LogisticRegression** (const arma::mat &predictors, const arma::vec &responses, const arma::mat &initialPoint, const double **lambda**=0)

  *Construct the **LogisticRegression*** (p. *438*) *class with the given labeled training data.*
- **LogisticRegression** (OptimizerType< **LogisticRegressionFunction** > &optimizer)

  *Construct the **LogisticRegression*** (p. *438*) *class with the given labeled training data.*

- **LogisticRegression** (const arma::vec &**parameters**, const double **lambda**=0)

  Construct a logistic regression model from the given parameters, without performing any training.
- double **ComputeAccuracy** (const arma::mat &predictors, const arma::vec &responses, const double decisionBoundary=0.5) const

  Compute the accuracy of the model on the given predictors and responses, optionally using the given decision boundary.
- double **ComputeError** (const arma::mat &predictors, const arma::vec &responses) const

  Compute the error of the model.
- const double & **Lambda** () const

  Return the lambda value for L2-regularization.
- double & **Lambda** ()

  Modify the lambda value for L2-regularization.
- const arma::vec & **Parameters** () const

  Return the parameters (the b vector).
- arma::vec & **Parameters** ()

  Modify the parameters (the b vector).
- void **Predict** (const arma::mat &predictors, arma::vec &responses, const double decisionBoundary=0.5) const

  Predict the responses to a given set of predictors.

Private Attributes

- double **lambda**

  L2-regularization penalty parameter.
- arma::vec **parameters**

  Vector of trained parameters.

21.97.1 Detailed Description

template<template< typename > class OptimizerType = mlpack::optimization::L_BFGS>class mlpack::regression::LogisticRegression< OptimizerType >

Definition at line 37 of file logistic_regression.hpp.

21.97.2 Constructor & Destructor Documentation

21.97.2.1 template<template< typename > class OptimizerType = mlpack::optimization::L_BFGS> mlpack::regression::LogisticRegression< OptimizerType >::LogisticRegression (const arma::mat & *predictors,* const arma::vec & *responses,* const double *lambda* = 0)

Construct the **LogisticRegression** (p. 438) class with the given labeled training data.

This will train the model. Optionally, specify lambda, which is the penalty parameter for L2-regularization. If not specified, it is set to 0, which results in standard (unregularized) logistic regression.

Parameters

predictors	Input training variables.
responses	Outputs resulting from input training variables.
lambda	L2-regularization parameter.

21.97.2.2 template<template< typename > class OptimizerType = mlpack::optimization::L_BFGS> mlpack::regression::LogisticRegression< OptimizerType >::LogisticRegression (const arma::mat & *predictors*, const arma::vec & *responses*, const arma::mat & *initialPoint*, const double *lambda* = 0)

Construct the **LogisticRegression** (p. 438) class with the given labeled training data.

This will train the model. Optionally, specify lambda, which is the penalty parameter for L2-regularization. If not specified, it is set to 0, which results in standard (unregularized) logistic regression.

Parameters

predictors	Input training variables.
responses	Outputs results from input training variables.
initialPoint	Initial model to train with.
lambda	L2-regularization parameter.

21.97.2.3 template<template< typename > class OptimizerType = mlpack::optimization::L_BFGS> mlpack::regression::LogisticRegression< OptimizerType >::LogisticRegression (OptimizerType< LogisticRegressionFunction > & *optimizer*)

Construct the **LogisticRegression** (p. 438) class with the given labeled training data.

This will train the model. This overload takes an already instantiated optimizer (which holds the **LogisticRegressionFunction** (p. 443) error function, which must also be instantiated), so that the optimizer can be configured before the training is run by this constructor. The predictors and responses and initial point are all taken from the error function contained in the optimizer.

Parameters

optimizer	Instantiated optimizer with instantiated error function.

21.97.2.4 template<template< typename > class OptimizerType = mlpack::optimization::L_BFGS> mlpack::regression::LogisticRegression< OptimizerType >::LogisticRegression (const arma::vec & *parameters*, const double *lambda* = 0)

Construct a logistic regression model from the given parameters, without performing any training.

The lambda parameter is used for the **ComputeAccuracy()** (p. 441) and **ComputeError()** (p. 441) functions; this constructor does not train the model itself.

Parameters

parameters	Parameters making up the model.
lambda	L2-regularization penalty parameter.

21.97.3 Member Function Documentation

21.97.3.1 template<template< typename > class OptimizerType = mlpack::optimization::L_BFGS> double mlpack::regression::LogisticRegression< OptimizerType >::ComputeAccuracy (const arma::mat & *predictors,* const arma::vec & *responses,* const double *decisionBoundary* = 0.5) const

Compute the accuracy of the model on the given predictors and responses, optionally using the given decision boundary.

The responses should be either 0 or 1. Logistic regression returns a value between 0 and 1. If the value is greater than the decision boundary, the response is taken to be 1; otherwise, it is 0. By default, the decision boundary is 0.5.

The accuracy is returned as a percentage, between 0 and 100.

Parameters

predictors	Input predictors.
responses	Vector of responses.
decision-Boundary	Decision boundary (default 0.5).

Returns

Percentage of responses that are predicted correctly.

21.97.3.2 template<template< typename > class OptimizerType = mlpack::optimization::L_BFGS> double mlpack::regression::LogisticRegression< OptimizerType >::ComputeError (const arma::mat & *predictors,* const arma::vec & *responses*) const

Compute the error of the model.

This returns the negative objective function of the logistic regression log-likelihood function. For the model to be optimal, the negative log-likelihood function should be minimized.

Parameters

predictors	Input predictors.
responses	Vector of responses.

21.97.3.3 template<template< typename > class OptimizerType = mlpack::optimization::L_BFGS> const double& mlpack::regression::LogisticRegression< OptimizerType >::Lambda ( ) const [inline]

Return the lambda value for L2-regularization.

Definition at line 100 of file logistic_regression.hpp.

References mlpack::regression::LogisticRegression< OptimizerType >::lambda.

21.97.3.4 template<template< typename > class OptimizerType = mlpack::optimization::L_BFGS> double& mlpack::regression::LogisticRegression< OptimizerType >::Lambda ( ) [inline]

Modify the lambda value for L2-regularization.

Definition at line 102 of file logistic_regression.hpp.

References mlpack::regression::LogisticRegression< OptimizerType >::lambda.

21.97.3.5 template<template< typename > class OptimizerType = mlpack::optimization::L_BFGS> const arma::vec& mlpack::regression::LogisticRegression< OptimizerType >::Parameters ( ) const [inline]

Return the parameters (the b vector).

Definition at line 95 of file logistic_regression.hpp.

References mlpack::regression::LogisticRegression< OptimizerType >::parameters.

21.97.3.6 template<template< typename > class OptimizerType = mlpack::optimization::L_BFGS> arma::vec& mlpack::regression::LogisticRegression< OptimizerType >::Parameters ( ) [inline]

Modify the parameters (the b vector).

Definition at line 97 of file logistic_regression.hpp.

References mlpack::regression::LogisticRegression< OptimizerType >::parameters.

21.97.3.7 template<template< typename > class OptimizerType = mlpack::optimization::L_BFGS> void mlpack::regression::LogisticRegression< OptimizerType >::Predict (const arma::mat & *predictors*, arma::vec & *responses*, const double *decisionBoundary* = 0.5) const

Predict the responses to a given set of predictors.

The responses will be either 0 or 1. Optionally, specify the decision boundary; logistic regression returns a value between 0 and 1. If the value is greater than the decision boundary, the response is taken to be 1; otherwise, it is 0. By default the decision boundary is 0.5.

Parameters

predictors	Input predictors.
responses	Vector to put output predictions of responses into.
decision-Boundary	Decision boundary (default 0.5).

21.97.4 Member Data Documentation

21.97.4.1 template<template< typename > class OptimizerType = mlpack::optimization::L_BFGS> double mlpack::regression::LogisticRegression< OptimizerType >::lambda [private]

L2-regularization penalty parameter.

Definition at line 152 of file logistic_regression.hpp.

Referenced by mlpack::regression::LogisticRegression< OptimizerType >::Lambda().

21.97.4.2 template<template< typename > class OptimizerType = mlpack::optimization::L_BFGS> arma::vec mlpack::regression::LogisticRegression< OptimizerType >::parameters [private]

Vector of trained parameters.

Definition at line 150 of file logistic_regression.hpp.

Referenced by mlpack::regression::LogisticRegression< OptimizerType >::Parameters().

The documentation for this class was generated from the following file:

- src/mlpack/methods/logistic_regression/**logistic_regression.hpp**

21.98 mlpack::regression::LogisticRegressionFunction Class Reference

The log-likelihood function for the logistic regression objective function.

Public Member Functions

- **LogisticRegressionFunction** (const arma::mat &**predictors**, const arma::vec &**responses**, const double **lambda**=0)
- **LogisticRegressionFunction** (const arma::mat &**predictors**, const arma::vec &**responses**, const arma::mat &**initialPoint**, const double **lambda**=0)
- double **Evaluate** (const arma::mat &parameters) const

  Evaluate the logistic regression log-likelihood function with the given parameters.
- double **Evaluate** (const arma::mat &parameters, const size_t i) const

  Evaluate the logistic regression log-likelihood function with the given parameters, but using only one data point.
- const arma::mat & **GetInitialPoint** () const

  Return the initial point for the optimization.
- void **Gradient** (const arma::mat &parameters, arma::mat &gradient) const

  Evaluate the gradient of the logistic regression log-likelihood function with the given parameters.
- void **Gradient** (const arma::mat &parameters, const size_t i, arma::mat &gradient) const

  Evaluate the gradient of the logistic regression log-likelihood function with the given parameters, and with respect to only one point in the dataset.
- const arma::mat & **InitialPoint** () const

  Return the initial point for the optimization.
- arma::mat & **InitialPoint** ()

  Modify the initial point for the optimization.
- const double & **Lambda** () const

  Return the regularization parameter (lambda).
- double & **Lambda** ()

  Modify the regularization parameter (lambda).
- size_t **NumFunctions** () const

  Return the number of separable functions (the number of predictor points).
- const arma::mat & **Predictors** () const

  Return the matrix of predictors.
- const arma::vec & **Responses** () const

  Return the vector of responses.

Private Attributes

- arma::mat **initialPoint**

  The initial point, from which to start the optimization.
- double **lambda**

  The regularization parameter for L2-regularization.
- const arma::mat & **predictors**

  The matrix of data points (predictors).
- const arma::vec & **responses**

  The vector of responses to the input data points.

21.98.1 Detailed Description

The log-likelihood function for the logistic regression objective function.

This is used by various mlpack optimizers to train a logistic regression model.

Definition at line 37 of file logistic_regression_function.hpp.

21.98.2 Constructor & Destructor Documentation

21.98.2.1 mlpack::regression::LogisticRegressionFunction::LogisticRegressionFunction (const arma::mat & *predictors,* const arma::vec & *responses,* const double *lambda* = 0)

21.98.2.2 mlpack::regression::LogisticRegressionFunction::LogisticRegressionFunction (const arma::mat & *predictors,* const arma::vec & *responses,* const arma::mat & *initialPoint,* const double *lambda* = 0)

21.98.3 Member Function Documentation

21.98.3.1 double mlpack::regression::LogisticRegressionFunction::Evaluate (const arma::mat & *parameters*) const

Evaluate the logistic regression log-likelihood function with the given parameters.

Note that if a point has 0 probability of being classified directly with the given parameters, then **Evaluate()** (p. 444) will return nan (this is kind of a corner case and should not happen for reasonable models).

The optimum (minimum) of this function is 0.0, and occurs when each point is classified correctly with very high probability.

Parameters

parameters	Vector of logistic regression parameters.

21.98.3.2 double mlpack::regression::LogisticRegressionFunction::Evaluate (const arma::mat & *parameters,* const size_t *i*) const

Evaluate the logistic regression log-likelihood function with the given parameters, but using only one data point.

This is useful for optimizers such as SGD, which require a separable objective function. Note that if the point has 0 probability of being classified correctly with the given parameters, then **Evaluate()** (p. 444) will return nan (this is kind of a corner case and should not happen for reasonable models).

The optimum (minimum) of this function is 0.0, and occurs when the point is classified correctly with very high probability.

Parameters

parameters	Vector of logistic regression parameters.
i	Index of point to use for objective function evaluation.

21.98.3.3 const arma::mat& mlpack::regression::LogisticRegressionFunction::GetInitialPoint ( ) const [inline]

Return the initial point for the optimization.

Definition at line 117 of file logistic_regression_function.hpp.

References initialPoint.

21.98.3.4 void mlpack::regression::LogisticRegressionFunction::Gradient (const arma::mat & *parameters,* arma::mat & *gradient*) const

Evaluate the gradient of the logistic regression log-likelihood function with the given parameters.

Parameters

parameters	Vector of logistic regression parameters.
gradient	Vector to output gradient into.

21.98.3.5 void mlpack::regression::LogisticRegressionFunction::Gradient (const arma::mat & *parameters,* const size_t *i,* arma::mat & *gradient*) const

Evaluate the gradient of the logistic regression log-likelihood function with the given parameters, and with respect to only one point in the dataset.

This is useful for optimizers such as SGD, which require a separable objective function.

Parameters

parameters	Vector of logistic regression parameters.
i	Index of points to use for objective function gradient evaluation.
gradient	Vector to output gradient into.

21.98.3.6 const arma::mat& mlpack::regression::LogisticRegressionFunction::InitialPoint ( ) const [inline]

Return the initial point for the optimization.

Definition at line 50 of file logistic_regression_function.hpp.

References initialPoint.

21.98.3.7 arma::mat& mlpack::regression::LogisticRegressionFunction::InitialPoint ( ) [inline]

Modify the initial point for the optimization.

Definition at line 52 of file logistic_regression_function.hpp.

References initialPoint.

21.98.3.8 const double& mlpack::regression::LogisticRegressionFunction::Lambda ( ) const [inline]

Return the regularization parameter (lambda).

Definition at line 55 of file logistic_regression_function.hpp.

References lambda.

21.98.3.9 double& mlpack::regression::LogisticRegressionFunction::Lambda ( ) [inline]

Modify the regularization parameter (lambda).

Definition at line 57 of file logistic_regression_function.hpp.

References lambda.

21.98.3.10 size_t mlpack::regression::LogisticRegressionFunction::NumFunctions ( ) const [inline]

Return the number of separable functions (the number of predictor points).

Definition at line 120 of file logistic_regression_function.hpp.

21.98.3.11 const arma::mat& mlpack::regression::LogisticRegressionFunction::Predictors ( ) const [inline]

Return the matrix of predictors.

Definition at line 60 of file logistic_regression_function.hpp.

References predictors.

21.98.3.12 const arma::vec& mlpack::regression::LogisticRegressionFunction::Responses ( ) const [inline]

Return the vector of responses.

Definition at line 62 of file logistic_regression_function.hpp.

References responses.

21.98.4 Member Data Documentation

21.98.4.1 arma::mat mlpack::regression::LogisticRegressionFunction::initialPoint [private]

The initial point, from which to start the optimization.

Definition at line 124 of file logistic_regression_function.hpp.

Referenced by GetInitialPoint(), and InitialPoint().

21.98.4.2 double mlpack::regression::LogisticRegressionFunction::lambda [private]

The regularization parameter for L2-regularization.

Definition at line 130 of file logistic_regression_function.hpp.

Referenced by Lambda().

21.98.4.3 const arma::mat& mlpack::regression::LogisticRegressionFunction::predictors [private]

The matrix of data points (predictors).

Definition at line 126 of file logistic_regression_function.hpp.

Referenced by Predictors().

21.98.4.4 const arma::vec& mlpack::regression::LogisticRegressionFunction::responses [private]

The vector of responses to the input data points.

Definition at line 128 of file logistic_regression_function.hpp.

Referenced by Responses().

The documentation for this class was generated from the following file:

- src/mlpack/methods/logistic_regression/**logistic_regression_function.hpp**

21.99 mlpack::sparse_coding::DataDependentRandomInitializer Class Reference

A data-dependent random dictionary initializer for **SparseCoding** (p. 450).

Static Public Member Functions

- static void **Initialize** (const arma::mat &data, const size_t atoms, arma::mat &dictionary)

  Initialize the dictionary by adding together three random observations from the data, and then normalizing the atom.

21.99.1 Detailed Description

A data-dependent random dictionary initializer for **SparseCoding** (p. 450).

This creates random dictionary atoms by adding three random observations from the data together, and then normalizing the atom.

Definition at line 35 of file data_dependent_random_initializer.hpp.

21.99.2 Member Function Documentation

21.99.2.1 static void mlpack::sparse_coding::DataDependentRandomInitializer::Initialize (const arma::mat & *data,* const size_t *atoms,* arma::mat & *dictionary*) [inline],[static]

Initialize the dictionary by adding together three random observations from the data, and then normalizing the atom.

This implementation is simple enough to be included with the definition.

Parameters

data	Dataset to initialize the dictionary with.
atoms	Number of atoms in dictionary.
dictionary	Dictionary to initialize.

Definition at line 47 of file data_dependent_random_initializer.hpp.

References mlpack::math::RandInt().

The documentation for this class was generated from the following file:

- src/mlpack/methods/sparse_coding/**data_dependent_random_initializer.hpp**

21.100 mlpack::sparse_coding::NothingInitializer Class Reference

A DictionaryInitializer for **SparseCoding** (p. 450) which does not initialize anything; it is useful for when the dictionary is already known and will be set with **SparseCoding::Dictionary()** (p. 453).

Static Public Member Functions

- static void **Initialize** (const arma::mat &, const size_t, arma::mat &)

  This function does not initialize the dictionary.

21.100.1 Detailed Description

A DictionaryInitializer for **SparseCoding** (p. 450) which does not initialize anything; it is useful for when the dictionary is already known and will be set with **SparseCoding::Dictionary()** (p. 453).

Definition at line 37 of file nothing_initializer.hpp.

21.100.2 Member Function Documentation

21.100.2.1 static void mlpack::sparse_coding::NothingInitializer::Initialize (const arma::mat & , const size_t , arma::mat &) [inline], [static]

This function does not initialize the dictionary.

This will cause problems for **SparseCoding** (p. 450) if the dictionary is not set manually before running the method.

Definition at line 45 of file nothing_initializer.hpp.

The documentation for this class was generated from the following file:

- src/mlpack/methods/sparse_coding/**nothing_initializer.hpp**

21.101 mlpack::sparse_coding::RandomInitializer Class Reference

A DictionaryInitializer for use with the **SparseCoding** (p. 450) class.

Static Public Member Functions

- static void **Initialize** (const arma::mat &data, const size_t atoms, arma::mat &dictionary)
  Initialize the dictionary randomly from a normal distribution, such that each atom has a norm of 1.

21.101.1 Detailed Description

A DictionaryInitializer for use with the **SparseCoding** (p. 450) class.

This provides a random, normally distributed dictionary, such that each atom has a norm of 1.

Definition at line 35 of file random_initializer.hpp.

21.101.2 Member Function Documentation

21.101.2.1 static void mlpack::sparse_coding::RandomInitializer::Initialize (const arma::mat & *data*, const size_t *atoms*, arma::mat & *dictionary*) [inline], [static]

Initialize the dictionary randomly from a normal distribution, such that each atom has a norm of 1.

This is simple enough to be included with the definition.

Parameters

data	Dataset to use for initialization.
atoms	Number of atoms (columns) in the dictionary.
dictionary	Dictionary to initialize.

Definition at line 47 of file random_initializer.hpp.

The documentation for this class was generated from the following file:

- src/mlpack/methods/sparse_coding/**random_initializer.hpp**

21.102 mlpack::sparse_coding::SparseCoding< DictionaryInitializer > Class Template Reference

An implementation of Sparse Coding with Dictionary Learning that achieves sparsity via an l1-norm regularizer on the codes (LASSO) or an (l1+l2)-norm regularizer on the codes (the Elastic Net).

Public Member Functions

- **SparseCoding** (const arma::mat &**data**, const size_t **atoms**, const double **lambda1**, const double **lambda2**=0)

  *Set the parameters to **SparseCoding** (p. 450).*
- const arma::mat & **Codes** () const

  Access the sparse codes.
- arma::mat & **Codes** ()

  Modify the sparse codes.
- const arma::mat & **Data** () const

  Access the data.
- const arma::mat & **Dictionary** () const

  Access the dictionary.
- arma::mat & **Dictionary** ()

  Modify the dictionary.
- void **Encode** (const size_t maxIterations=0, const double objTolerance=0.01, const double newtonTolerance=1e-6)

  Run Sparse Coding with Dictionary Learning.
- double **Objective** () const

  Compute the objective function.
- void **OptimizeCode** ()

  Sparse code each point via LARS.
- double **OptimizeDictionary** (const arma::uvec &adjacencies, const double newtonTolerance=1e-6)

  Learn dictionary via Newton method based on Lagrange dual.
- void **ProjectDictionary** ()

  Project each atom of the dictionary back onto the unit ball, if necessary.

Private Attributes

- size_t **atoms**

  Number of atoms.
- arma::mat **codes**

  Sparse codes (columns are points).
- const arma::mat & **data**

  Data matrix (columns are points).
- arma::mat **dictionary**

  Dictionary (columns are atoms).
- double **lambda1**

  l1 regularization term.
- double **lambda2**

  l2 regularization term.

21.102.1 Detailed Description

template<typename DictionaryInitializer = DataDependentRandomInitializer>class mlpack::sparse_coding::SparseCoding< DictionaryInitializer >

An implementation of Sparse Coding with Dictionary Learning that achieves sparsity via an l1-norm regularizer on the codes (LASSO) or an (l1+l2)-norm regularizer on the codes (the Elastic Net).

Let d be the number of dimensions in the original space, m the number of training points, and k the number of atoms in the dictionary (the dimension of the learned feature space). The training data X is a d-by-m matrix where each column is a point and each row is a dimension. The dictionary D is a d-by-k matrix, and the sparse codes matrix Z is a k-by-m matrix. This program seeks to minimize the objective:

$$\min_{D,Z} 0.5||X - DZ||_F^2 + \lambda_1 \sum_{i=1}^{m} ||Z_i||_1 + 0.5\lambda_2 \sum_{i=1}^{m} ||Z_i||_2^2$$

subject to $||D_j||_2 <= 1$ for $1 <= j <= k$ where typically $lambda_1 > 0$ and $lambda_2 = 0$.

This problem is solved by an algorithm that alternates between a dictionary learning step and a sparse coding step. The dictionary learning step updates the dictionary D using a Newton method based on the Lagrange dual (see the paper below for details). The sparse coding step involves solving a large number of sparse linear regression problems; this can be done efficiently using LARS, an algorithm that can solve the LASSO or the Elastic Net (papers below).

Here are those papers:

```
@incollection{lee2007efficient,
  title = {Efficient sparse coding algorithms},
  author = {Honglak Lee and Alexis Battle and Rajat Raina and Andrew Y. Ng},
  booktitle = {Advances in Neural Information Processing Systems 19},
  editor = {B. Sch\"{o}lkopf and J. Platt and T. Hoffman},
  publisher = {MIT Press},
  address = {Cambridge, MA},
  pages = {801--808},
  year = {2007}
}


@article{efron2004least,
  title={Least angle regression},
  author={Efron, B. and Hastie, T. and Johnstone, I. and Tibshirani, R.},
  journal={The Annals of statistics},
  volume={32},
  number={2},
  pages={407--499},
```

```
  year={2004},
  publisher={Institute of Mathematical Statistics}
}


@article{zou2005regularization,
  title={Regularization and variable selection via the elastic net},
  author={Zou, H. and Hastie, T.},
  journal={Journal of the Royal Statistical Society Series B},
  volume={67},
  number={2},
  pages={301--320},
  year={2005},
  publisher={Royal Statistical Society}
}
```

Before the method is run, the dictionary is initialized using the DictionaryInitializationPolicy class. Possible choices include the **RandomInitializer** (p. 449), which provides an entirely random dictionary, the **DataDependentRandomInitializer** (p. 448), which provides a random dictionary based loosely on characteristics of the dataset, and the **NothingInitializer** (p. 448), which does not initialize the dictionary – instead, the user should set the dictionary using the **Dictionary()** (p. 453) mutator method.

Template Parameters

DictionaryInitializationPolicy	The class to use to initialize the dictionary; must have 'void Initialize(const arma::mat& data, arma::mat& dictionary)' function.

Definition at line 119 of file sparse_coding.hpp.

21.102.2 Constructor & Destructor Documentation

21.102.2.1 template<typename DictionaryInitializer = DataDependentRandomInitializer> mlpack::sparse_coding::SparseCoding< DictionaryInitializer >::SparseCoding (const arma::mat & *data,* const size_t *atoms,* const double *lambda1,* const double *lambda2* = 0)

Set the parameters to **SparseCoding** (p. 450).

lambda2 defaults to 0.

Parameters

data	Data matrix
atoms	Number of atoms in dictionary
lambda1	Regularization parameter for l1-norm penalty
lambda2	Regularization parameter for l2-norm penalty

21.102.3 Member Function Documentation

21.102.3.1 template<typename DictionaryInitializer = DataDependentRandomInitializer> const arma::mat& mlpack::sparse_coding::SparseCoding< DictionaryInitializer >::Codes ( ) const [inline]

Access the sparse codes.

Definition at line 187 of file sparse_coding.hpp.

References mlpack::sparse_coding::SparseCoding< DictionaryInitializer >::codes.

21.102.3.2 template<typename DictionaryInitializer = DataDependentRandomInitializer> arma::mat& mlpack::sparse_coding::SparseCoding< DictionaryInitializer >::Codes ( ) [inline]

Modify the sparse codes.

Definition at line 189 of file sparse_coding.hpp.

References mlpack::sparse_coding::SparseCoding< DictionaryInitializer >::codes.

21.102.3.3 template<typename DictionaryInitializer = DataDependentRandomInitializer> const arma::mat& mlpack::sparse_coding::SparseCoding< DictionaryInitializer >::Data ( ) const [inline]

Access the data.

Definition at line 179 of file sparse_coding.hpp.

References mlpack::sparse_coding::SparseCoding< DictionaryInitializer >::data.

21.102.3.4 template<typename DictionaryInitializer = DataDependentRandomInitializer> const arma::mat& mlpack::sparse_coding::SparseCoding< DictionaryInitializer >::Dictionary ( ) const [inline]

Access the dictionary.

Definition at line 182 of file sparse_coding.hpp.

References mlpack::sparse_coding::SparseCoding< DictionaryInitializer >::dictionary.

21.102.3.5 template<typename DictionaryInitializer = DataDependentRandomInitializer> arma::mat& mlpack::sparse_coding::SparseCoding< DictionaryInitializer >::Dictionary ( ) [inline]

Modify the dictionary.

Definition at line 184 of file sparse_coding.hpp.

References mlpack::sparse_coding::SparseCoding< DictionaryInitializer >::dictionary.

21.102.3.6 template<typename DictionaryInitializer = DataDependentRandomInitializer> void mlpack::sparse_coding::SparseCoding< DictionaryInitializer >::Encode (const size_t *maxIterations* = 0, const double *objTolerance* = 0.01, const double *newtonTolerance* = 1e-6)

Run Sparse Coding with Dictionary Learning.

Parameters

maxIterations	Maximum number of iterations to run algorithm. If 0, the algorithm will run until convergence (or forever).
objTolerance	Tolerance for objective function. When an iteration of the algorithm produces an improvement smaller than this, the algorithm will terminate.
newtonTolerance	Tolerance for the Newton's method dictionary optimization step.

21.102.3.7 template<typename DictionaryInitializer = DataDependentRandomInitializer> double mlpack::sparse_coding::SparseCoding< DictionaryInitializer >::Objective ( ) const

Compute the objective function.

21.102.3.8 template<typename DictionaryInitializer = DataDependentRandomInitializer> void mlpack::sparse_coding::SparseCoding< DictionaryInitializer >::OptimizeCode ( )

Sparse code each point via LARS.

21.102.3.9 template<typename DictionaryInitializer = DataDependentRandomInitializer> double mlpack::sparse_coding::SparseCoding< DictionaryInitializer >::OptimizeDictionary (const arma::uvec & *adjacencies,* const double *newtonTolerance* = 1e-6)

Learn dictionary via Newton method based on Lagrange dual.

Parameters

adjacencies	Indices of entries (unrolled column by column) of the coding matrix Z that are non-zero (the adjacency matrix for the bipartite graph of points and atoms).
newtonTolerance	Tolerance of the Newton's method optimizer.

Returns

the norm of the gradient of the Lagrange dual with respect to the dual variables

21.102.3.10 template<typename DictionaryInitializer = DataDependentRandomInitializer> void mlpack::sparse_coding::SparseCoding< DictionaryInitializer >::ProjectDictionary ( )

Project each atom of the dictionary back onto the unit ball, if necessary.

21.102.4 Member Data Documentation

21.102.4.1 template<typename DictionaryInitializer = DataDependentRandomInitializer> size_t mlpack::sparse_coding::SparseCoding< DictionaryInitializer >::atoms [private]

Number of atoms.

Definition at line 193 of file sparse_coding.hpp.

21.102.4.2 template<typename DictionaryInitializer = DataDependentRandomInitializer> arma::mat mlpack::sparse_coding::SparseCoding< DictionaryInitializer >::codes [private]

Sparse codes (columns are points).

Definition at line 202 of file sparse_coding.hpp.

Referenced by mlpack::sparse_coding::SparseCoding< DictionaryInitializer >::Codes().

21.102.4.3 template<typename DictionaryInitializer = DataDependentRandomInitializer> const arma::mat& mlpack::sparse_coding::SparseCoding< DictionaryInitializer >::data [private]

Data matrix (columns are points).

Definition at line 196 of file sparse_coding.hpp.

Referenced by mlpack::sparse_coding::SparseCoding< DictionaryInitializer >::Data().

21.102.4.4 template<typename DictionaryInitializer = DataDependentRandomInitializer> arma::mat mlpack::sparse_coding::SparseCoding< DictionaryInitializer >::dictionary [private]

Dictionary (columns are atoms).

Definition at line 199 of file sparse_coding.hpp.

Referenced by mlpack::sparse_coding::SparseCoding< DictionaryInitializer >::Dictionary().

21.102.4.5 template<typename DictionaryInitializer = DataDependentRandomInitializer> double mlpack::sparse_coding::SparseCoding< DictionaryInitializer >::lambda1 [private]

l1 regularization term.

Definition at line 205 of file sparse_coding.hpp.

21.102.4.6 template<typename DictionaryInitializer = DataDependentRandomInitializer> double mlpack::sparse_coding::SparseCoding< DictionaryInitializer >::lambda2 [private]

l2 regularization term.

Definition at line 208 of file sparse_coding.hpp.

The documentation for this class was generated from the following file:

- src/mlpack/methods/sparse_coding/**sparse_coding.hpp**

21.103 mlpack::Timer Class Reference

The timer class provides a way for MLPACK methods to be timed.

Static Public Member Functions

- static timeval **Get** (const std::string &name)

  Get the value of the given timer.
- static void **Start** (const std::string &name)

  Start the given timer.
- static void **Stop** (const std::string &name)

  Stop the given timer.

21.103.1 Detailed Description

The timer class provides a way for MLPACK methods to be timed.

The three methods contained in this class allow a named timer to be started and stopped, and its value to be obtained.

Definition at line 73 of file timers.hpp.

21.103.2 Member Function Documentation

21.103.2.1 static timeval mlpack::Timer::Get (const std::string & *name*) [static]

Get the value of the given timer.

Parameters

name	Name of timer to return value of.

21.103.2.2 static void mlpack::Timer::Start (const std::string & *name*) [static]

Start the given timer.

If a timer is started, then stopped, then re-started, then re-stopped, the final value of the timer is the length of both runs – that is, MLPACK timers are additive for each time they are run, and do not reset.

Note

Undefined behavior will occur if a timer is started twice.

Parameters

name	Name of timer to be started.

21.103.2.3 static void mlpack::Timer::Stop (const std::string & *name*) [static]

Stop the given timer.

Note

Undefined behavior will occur if a timer is started twice.

Parameters

name	Name of timer to be stopped.

The documentation for this class was generated from the following file:

- src/mlpack/core/util/**timers.hpp**

21.104 mlpack::Timers Class Reference

Public Member Functions

- **Timers** ()

  Nothing to do for the constructor.
- std::map< std::string, timeval > & **GetAllTimers** ()

  Returns a copy of all the timers used via this interface.
- timeval **GetTimer** (const std::string &timerName)

  Returns a copy of the timer specified.
- void **PrintTimer** (const std::string &timerName)

  Prints the specified timer.
- void **StartTimer** (const std::string &timerName)

  ∗ *Initializes a timer, available like a normal value specified on* ∗ *the command line.*
- void **StopTimer** (const std::string &timerName)

  ∗ *Halts the timer, and replaces it's value with* ∗ *the delta time from it's start* ∗ ∗.

Private Member Functions

- void **FileTimeToTimeVal** (timeval *tv)
- void **GetTime** (timeval *tv)

Private Attributes

- std::map< std::string, timeval > **timers**

21.104.1 Detailed Description

Definition at line 105 of file timers.hpp.

21.104.2 Constructor & Destructor Documentation

21.104.2.1 mlpack::Timers::Timers ( ) [inline]

Nothing to do for the constructor.

Definition at line 109 of file timers.hpp.

21.104.3 Member Function Documentation

21.104.3.1 void mlpack::Timers::FileTimeToTimeVal (timeval * *tv*) [private]

21.104.3.2 std::map<std::string, timeval>& mlpack::Timers::GetAllTimers ( )

Returns a copy of all the timers used via this interface.

21.104.3.3 void mlpack::Timers::GetTime (timeval * *tv*) [private]

21.104.3.4 timeval mlpack::Timers::GetTimer (const std::string & *timerName*)

Returns a copy of the timer specified.

Parameters

timerName	The name of the timer in question.

21.104.3.5 void mlpack::Timers::PrintTimer (const std::string & *timerName*)

Prints the specified timer.

If it took longer than a minute to complete the timer will be displayed in days, hours, and minutes as well.

Parameters

timerName	The name of the timer in question.

21.104.3.6 void mlpack::Timers::StartTimer (const std::string & *timerName*)

∗ Initializes a timer, available like a normal value specified on ∗ the command line.

Timers (p. 457) are of type timeval. If a timer is started, then stopped, then re-started, then stopped, the final timer value will be the length of both runs of the timer. ∗ ∗

Parameters

timerName	The name of the timer in question.

21.104.3.7 void mlpack::Timers::StopTimer (const std::string & *timerName*)

∗ Halts the timer, and replaces it's value with ∗ the delta time from it's start ∗ ∗.

Parameters

timerName	The name of the timer in question.

21.104.4 Member Data Documentation

21.104.4.1 std::map<std::string, timeval> mlpack::Timers::timers [private]

Definition at line 150 of file timers.hpp.

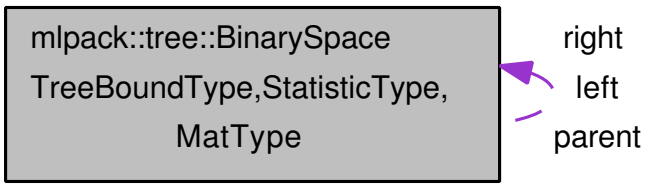
The documentation for this class was generated from the following file:

- src/mlpack/core/util/**timers.hpp**

21.105 mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType > Class Template Reference

A binary space partitioning tree, such as a KD-tree or a ball tree.

Collaboration diagram for mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >:



Classes

- class **DualTreeTraverser**

A dual-tree traverser for binary space trees; see dual_tree_traverser.hpp.

- class **SingleTreeTraverser**

  A single-tree traverser for binary space trees; see single_tree_traverser.hpp for implementation.

Public Types

- typedef MatType **Mat**

  So other classes can use TreeType::Mat.

Public Member Functions

- **BinarySpaceTree** (MatType &data, const size_t **leafSize**=20)

  Construct this as the root node of a binary space tree using the given dataset.
- **BinarySpaceTree** (MatType &data, std::vector< size_t > &oldFromNew, const size_t **leafSize**=20)

  Construct this as the root node of a binary space tree using the given dataset.
- **BinarySpaceTree** (MatType &data, std::vector< size_t > &oldFromNew, std::vector< size_t > &newFromOld, const size_t **leafSize**=20)

  Construct this as the root node of a binary space tree using the given dataset.
- **BinarySpaceTree** (MatType &data, const size_t **begin**, const size_t **count**, **BinarySpaceTree** *__parent__=NULL, const size_t **leafSize**=20)

  Construct this node on a subset of the given matrix, starting at column begin and using count points.
- **BinarySpaceTree** (MatType &data, const size_t **begin**, const size_t **count**, std::vector< size_t > &oldFromNew, **BinarySpaceTree** *__parent__=NULL, const size_t **leafSize**=20)

  Construct this node on a subset of the given matrix, starting at column begin_in and using count_in points.
- **BinarySpaceTree** (MatType &data, const size_t **begin**, const size_t **count**, std::vector< size_t > &oldFromNew, std::vector< size_t > &newFromOld, **BinarySpaceTree** *__parent__=NULL, const size_t **leafSize**=20)

  Construct this node on a subset of the given matrix, starting at column begin_in and using count_in points.
- **BinarySpaceTree** (const **BinarySpaceTree** &other)

  Create a binary space tree by copying the other tree.
- ~**BinarySpaceTree** ()

  Deletes this node, deallocating the memory for the children and calling their destructors in turn.
- size_t **Begin** () const

  Return the index of the beginning point of this subset.
- size_t & **Begin** ()

  Modify the index of the beginning point of this subset.
- const BoundType & **Bound** () const

  Return the bound object for this node.
- BoundType & **Bound** ()

  Return the bound object for this node.
- void **Centroid** (arma::vec &centroid)

  Get the centroid of the node and store it in the given vector.
- **BinarySpaceTree** & **Child** (const size_t child) const

  Return the specified child (0 will be left, 1 will be right).
- size_t **Count** () const

  Return the number of points in this subset.
- size_t & **Count** ()

  Modify the number of points in this subset.

- const arma::mat & **Dataset** () const

  Get the dataset which the tree is built on.
- arma::mat & **Dataset** ()

  Modify the dataset which the tree is built on. Be careful!
- size_t **Descendant** (const size_t index) const

  Return the index (with reference to the dataset) of a particular descendant of this node.
- size_t **End** () const

  Gets the index one beyond the last index in the subset.
- size_t **ExtendTree** (const size_t level)

  Fills the tree to the specified level.
- const **BinarySpaceTree** * **FindByBeginCount** (size_t **begin**, size_t **count**) const

  Find a node in this tree by its begin and count (const).
- **BinarySpaceTree** * **FindByBeginCount** (size_t **begin**, size_t **count**)

  Find a node in this tree by its begin and count.
- double **FurthestDescendantDistance** () const

  Return the furthest possible descendant distance.
- double **FurthestPointDistance** () const

  Return the furthest distance to a point held in this node.
- size_t **GetSplitDimension** () const

  Returns the dimension this parent's children are split on.
- bool **IsLeaf** () const

  Return whether or not this node is a leaf (true if it has no children).
- size_t **LeafSize** () const

  Return the leaf size.
- size_t & **LeafSize** ()

  Modify the leaf size.
- **BinarySpaceTree** * **Left** () const

  Gets the left child of this node.
- **BinarySpaceTree** *& **Left** ()

  Modify the left child of this node.
- double **MaxDistance** (const **BinarySpaceTree** *other) const

  Return the maximum distance to another node.
- double **MaxDistance** (const arma::vec &point) const

  Return the maximum distance to another point.
- BoundType::MetricType **Metric** () const

  Get the metric which the tree uses.
- double **MinDistance** (const **BinarySpaceTree** *other) const

  Return the minimum distance to another node.
- double **MinDistance** (const arma::vec &point) const

  Return the minimum distance to another point.
- size_t **NumChildren** () const

  Return the number of children in this node.
- size_t **NumDescendants** () const

  Return the number of descendants of this node.
- size_t **NumPoints** () const

  Return the number of points in this node (0 if not a leaf).
- **BinarySpaceTree** * **Parent** () const

Gets the parent of this node.

- **BinarySpaceTree** *& **Parent** ()

  Modify the parent of this node.
- size_t **Point** (const size_t index) const

  Return the index (with reference to the dataset) of a particular point in this node.
- **math::Range RangeDistance** (const **BinarySpaceTree** *other) const

  Return the minimum and maximum distance to another node.
- **math::Range RangeDistance** (const arma::vec &point) const

  Return the minimum and maximum distance to another point.
- **BinarySpaceTree** * **Right** () const

  Gets the right child of this node.
- **BinarySpaceTree** *& **Right** ()

  Modify the right child of this node.
- size_t **SplitDimension** () const

  Get the split dimension for this node.
- size_t & **SplitDimension** ()

  Modify the split dimension for this node.
- const StatisticType & **Stat** () const

  Return the statistic object for this node.
- StatisticType & **Stat** ()

  Return the statistic object for this node.
- std::string **ToString** () const

  Returns a string representation of this object.
- size_t **TreeDepth** () const

  Obtains the number of levels below this node in the tree, starting with this.
- size_t **TreeSize** () const

  Obtains the number of nodes in the tree, starting with this.

Static Public Member Functions

- static bool **HasSelfChildren** ()

  Returns false: this tree type does not have self children.

Private Member Functions

- **BinarySpaceTree** (const size_t **begin**, const size_t **count**, BoundType **bound**, StatisticType **stat**, const int **leafSize**=20)

  Private copy constructor, available only to fill (pad) the tree to a specified level.
- **BinarySpaceTree** * **CopyMe** ()
- size_t **GetSplitIndex** (MatType &data, int splitDim, double splitVal)

  Find the index to split on for this node, given that we are splitting in the given split dimension on the specified split value.
- size_t **GetSplitIndex** (MatType &data, int splitDim, double splitVal, std::vector< size_t > &oldFromNew)

  Find the index to split on for this node, given that we are splitting in the given split dimension on the specified split value.
- void **SplitNode** (MatType &data)

  Splits the current node, assigning its left and right children recursively.
- void **SplitNode** (MatType &data, std::vector< size_t > &oldFromNew)

  Splits the current node, assigning its left and right children recursively.

Private Attributes

- size_t **begin**

  The index of the first point in the dataset contained in this node (and its children).

- BoundType **bound**

  The bound object for this node.

- size_t **count**

  The number of points of the dataset contained in this node (and its children).

- MatType & **dataset**

  The dataset.

- double **furthestDescendantDistance**

  The distance to the furthest descendant, cached to speed things up.

- size_t **leafSize**

  The leaf size.

- **BinarySpaceTree** * **left**

  The left child node.

- **BinarySpaceTree** * **parent**

  The parent node (NULL if this is the root of the tree).

- **BinarySpaceTree** * **right**

  The right child node.

- size_t **splitDimension**

  The dimension this node split on if it is a parent.

- StatisticType **stat**

  Any extra data contained in the node.

21.105.1 Detailed Description

template<typename BoundType, typename StatisticType = EmptyStatistic, typename MatType = arma::mat>class mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >

A binary space partitioning tree, such as a KD-tree or a ball tree.

Once the bound and type of dataset is defined, the tree will construct itself. Call the constructor with the dataset to build the tree on, and the entire tree will be built.

This particular tree does not allow growth, so you cannot add or delete nodes from it. If you need to add or delete a node, the better procedure is to rebuild the tree entirely.

This tree does take one parameter, which is the leaf size to be used.

Template Parameters

BoundType	The bound used for each node. The valid types of bounds and the necessary skeleton interface for this class can be found in bounds/.
StatisticType	Extra data contained in the node. See **statistic.hpp** (p. 620) for the necessary skeleton interface.

Definition at line 52 of file binary_space_tree.hpp.

21.105.2 Member Typedef Documentation

21.105.2.1 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> typedef MatType mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::Mat

So other classes can use TreeType::Mat.

Definition at line 82 of file binary_space_tree.hpp.

21.105.3 Constructor & Destructor Documentation

21.105.3.1 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::BinarySpaceTree (MatType & *data,* const size_t *leafSize* = 20)

Construct this as the root node of a binary space tree using the given dataset.

This will modify the ordering of the points in the dataset!

Parameters

data	Dataset to create tree from. This will be modified!
leafSize	Size of each leaf in the tree.

Referenced by mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::CopyMe().

21.105.3.2 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::BinarySpaceTree (MatType & *data,* std::vector< size_t > & *oldFromNew,* const size_t *leafSize* = 20)

Construct this as the root node of a binary space tree using the given dataset.

This will modify the ordering of points in the dataset! A mapping of the old point indices to the new point indices is filled.

Parameters

data	Dataset to create tree from. This will be modified!
oldFromNew	Vector which will be filled with the old positions for each new point.
leafSize	Size of each leaf in the tree.

21.105.3.3 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::BinarySpaceTree (MatType & *data,* std::vector< size_t > & *oldFromNew,* std::vector< size_t > & *newFromOld,* const size_t *leafSize* = 20)

Construct this as the root node of a binary space tree using the given dataset.

This will modify the ordering of points in the dataset! A mapping of the old point indices to the new point indices is filled, as well as a mapping of the new point indices to the old point indices.

Parameters

data	Dataset to create tree from. This will be modified!

oldFromNew	Vector which will be filled with the old positions for each new point.
newFromOld	Vector which will be filled with the new positions for each old point.
leafSize	Size of each leaf in the tree.

21.105.3.4 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::BinarySpaceTree (MatType & *data,* const size_t *begin,* const size_t *count,* BinarySpaceTree< BoundType, StatisticType, MatType > * *parent* = NULL, const size_t *leafSize* = 20)

Construct this node on a subset of the given matrix, starting at column begin and using count points.

The ordering of that subset of points will be modified! This is used for recursive tree-building by the other constructors which don't specify point indices.

Parameters

data	Dataset to create tree from. This will be modified!
begin	Index of point to start tree construction with.
count	Number of points to use to construct tree.
leafSize	Size of each leaf in the tree.

21.105.3.5 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::BinarySpaceTree (MatType & *data,* const size_t *begin,* const size_t *count,* std::vector< size_t > & *oldFromNew,* BinarySpaceTree< BoundType, StatisticType, MatType > * *parent* = NULL, const size_t *leafSize* = 20)

Construct this node on a subset of the given matrix, starting at column begin_in and using count_in points.

The ordering of that subset of points will be modified! This is used for recursive tree-building by the other constructors which don't specify point indices.

A mapping of the old point indices to the new point indices is filled, but it is expected that the vector is already allocated with size greater than or equal to (begin_in + count_in), and if that is not true, invalid memory reads (and writes) will occur.

Parameters

data	Dataset to create tree from. This will be modified!
begin	Index of point to start tree construction with.
count	Number of points to use to construct tree.
oldFromNew	Vector which will be filled with the old positions for each new point.
leafSize	Size of each leaf in the tree.

21.105.3.6 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::BinarySpaceTree (MatType & *data,* const size_t *begin,* const size_t *count,* std::vector< size_t > & *oldFromNew,* std::vector< size_t > & *newFromOld,* BinarySpaceTree< BoundType, StatisticType, MatType > * *parent* = NULL, const size_t *leafSize* = 20)

Construct this node on a subset of the given matrix, starting at column begin_in and using count_in points.

The ordering of that subset of points will be modified! This is used for recursive tree-building by the other constructors which don't specify point indices.

A mapping of the old point indices to the new point indices is filled, as well as a mapping of the new point indices to

the old point indices. It is expected that the vector is already allocated with size greater than or equal to (begin_in + count_in), and if that is not true, invalid memory reads (and writes) will occur.

Parameters

data	Dataset to create tree from. This will be modified!
begin	Index of point to start tree construction with.
count	Number of points to use to construct tree.
oldFromNew	Vector which will be filled with the old positions for each new point.
newFromOld	Vector which will be filled with the new positions for each old point.
leafSize	Size of each leaf in the tree.

21.105.3.7 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::BinarySpaceTree (const BinarySpaceTree< BoundType, StatisticType, MatType > & *other*)

Create a binary space tree by copying the other tree.

Be careful! This can take a long time and use a lot of memory.

Parameters

other	Tree to be replicated.

21.105.3.8 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::~BinarySpaceTree ( )

Deletes this node, deallocating the memory for the children and calling their destructors in turn.

This will invalidate any pointers or references to any nodes which are children of this one.

21.105.3.9 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::BinarySpaceTree (const size_t *begin,* const size_t *count,* BoundType *bound,* StatisticType *stat,* const int *leafSize* = 20) [inline], [private]

Private copy constructor, available only to fill (pad) the tree to a specified level.

Definition at line 429 of file binary_space_tree.hpp.

21.105.4 Member Function Documentation

21.105.4.1 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> size_t mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::Begin ( ) const [inline]

Return the index of the beginning point of this subset.

Definition at line 407 of file binary_space_tree.hpp.

References mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::begin.

21.105.4.2 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> size_t& mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::Begin ( ) [inline]

Modify the index of the beginning point of this subset.

Definition at line 409 of file binary_space_tree.hpp.

References mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::begin.

21.105.4.3 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> const BoundType& mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::Bound ( ) const [inline]

Return the bound object for this node.

Definition at line 248 of file binary_space_tree.hpp.

References mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::bound.

Referenced by mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::MaxDistance(), mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::MinDistance(), and mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::RangeDistance().

21.105.4.4 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> BoundType& mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::Bound ( ) [inline]

Return the bound object for this node.

Definition at line 250 of file binary_space_tree.hpp.

References mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::bound.

21.105.4.5 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> void mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::Centroid (arma::vec & *centroid*) [inline]

Get the centroid of the node and store it in the given vector.

Definition at line 297 of file binary_space_tree.hpp.

References mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::bound.

21.105.4.6 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> BinarySpaceTree& mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::Child (const size_t *child*) const

Return the specified child (0 will be left, 1 will be right).

If the index is greater than 1, this will return the right child.

Parameters

child	Index of child to return.

21.105.4.7 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> BinarySpaceTree* mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::CopyMe ( ) [inline],[private]

Definition at line 442 of file binary_space_tree.hpp.

References mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::begin, mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::BinarySpaceTree(), mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::bound, mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::count, mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::leafSize, and mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::stat.

21.105.4.8 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> size_t mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::Count ( ) const [inline]

Return the number of points in this subset.

Definition at line 417 of file binary_space_tree.hpp.

References mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::count.

21.105.4.9 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> size_t& mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::Count ( ) [inline]

Modify the number of points in this subset.

Definition at line 419 of file binary_space_tree.hpp.

References mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::count.

21.105.4.10 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> const arma::mat& mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::Dataset ( ) const [inline]

Get the dataset which the tree is built on.

Definition at line 289 of file binary_space_tree.hpp.

References mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::dataset.

21.105.4.11 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> arma::mat& mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::Dataset ( ) [inline]

Modify the dataset which the tree is built on. Be careful!

Definition at line 291 of file binary_space_tree.hpp.

References mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::dataset.

21.105.4.12 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> size_t mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::Descendant (const size_t *index*) const

Return the index (with reference to the dataset) of a particular descendant of this node.

The index should be greater than zero but less than the number of descendants.

Parameters

index	Index of the descendant.

21.105.4.13 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> size_t mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::End ( ) const

Gets the index one beyond the last index in the subset.

21.105.4.14 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> size_t mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::ExtendTree (const size_t *level*)

Fills the tree to the specified level.

21.105.4.15 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> const BinarySpaceTree* mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::FindByBeginCount (size_t *begin,* size_t *count*) const

Find a node in this tree by its begin and count (const).

Every node is uniquely identified by these two numbers. This is useful for communicating position over the network, when pointers would be invalid.

Parameters

begin	The **begin()** (p. 476) of the node to find.
count	The **count()** (p. 476) of the node to find.

Returns

The found node, or NULL if not found.

21.105.4.16 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> BinarySpaceTree* mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::FindByBeginCount (size_t *begin,* size_t *count*)

Find a node in this tree by its begin and count.

Every node is uniquely identified by these two numbers. This is useful for communicating position over the network, when pointers would be invalid.

Parameters

begin	The **begin()** (p. 476) of the node to find.
count	The **count()** (p. 476) of the node to find.

Returns

The found node, or NULL if not found.

21.105.4.17 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> double mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::FurthestDescendantDistance ( ) const

Return the furthest possible descendant distance.

This returns the maximum distance from the centroid to the edge of the bound and not the empirical quantity which is the actual furthest descendant distance. So the actual furthest descendant distance may be less than what this method returns (but it will never be greater than this).

21.105.4.18 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> double mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::FurthestPointDistance ( ) const

Return the furthest distance to a point held in this node.

If this is not a leaf node, then the distance is 0 because the node holds no points.

21.105.4.19 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> size_t mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::GetSplitDimension ( ) const

Returns the dimension this parent's children are split on.

21.105.4.20 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> size_t mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::GetSplitIndex (MatType & *data,* int *splitDim,* double *splitVal*) [private]

Find the index to split on for this node, given that we are splitting in the given split dimension on the specified split value.

Parameters

data	Dataset which we are using.
splitDim	Dimension of dataset to split on.
splitVal	Value to split on, in the given split dimension.

21.105.4.21 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> size_t mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::GetSplitIndex (MatType & *data,* int *splitDim,* double *splitVal,* std::vector< size_t > & *oldFromNew*) [private]

Find the index to split on for this node, given that we are splitting in the given split dimension on the specified split value.

Also returns a list of the changed indices.

Parameters

data	Dataset which we are using.
splitDim	Dimension of dataset to split on.
splitVal	Value to split on, in the given split dimension.
oldFromNew	Vector holding permuted indices.

21.105.4.22 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> static bool mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::HasSelfChildren ( ) [inline], [static]

Returns false: this tree type does not have self children.

Definition at line 422 of file binary_space_tree.hpp.

21.105.4.23 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> bool mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::IsLeaf ( ) const

Return whether or not this node is a leaf (true if it has no children).

21.105.4.24 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> size_t mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::LeafSize ( ) const [inline]

Return the leaf size.

Definition at line 261 of file binary_space_tree.hpp.

References mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::leafSize.

21.105.4.25 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> size_t& mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::LeafSize ( ) [inline]

Modify the leaf size.

Definition at line 263 of file binary_space_tree.hpp.

References mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::leafSize.

21.105.4.26 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> BinarySpaceTree* mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::Left ( ) const [inline]

Gets the left child of this node.

Definition at line 269 of file binary_space_tree.hpp.

References mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::left.

21.105.4.27 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> BinarySpaceTree*& mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::Left ( ) [inline]

Modify the left child of this node.

Definition at line 271 of file binary_space_tree.hpp.

References mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::left.

21.105.4.28 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> double **mlpack::tree::BinarySpaceTree**< BoundType, StatisticType, MatType >::MaxDistance (const **BinarySpaceTree**< BoundType, StatisticType, MatType > * *other*) const [inline]

Return the maximum distance to another node.

Definition at line 361 of file binary_space_tree.hpp.

References mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::bound, and mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::Bound().

21.105.4.29 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> double **mlpack::tree::BinarySpaceTree**< BoundType, StatisticType, MatType >::MaxDistance (const arma::vec & *point*) const [inline]

Return the maximum distance to another point.

Definition at line 379 of file binary_space_tree.hpp.

References mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::bound.

21.105.4.30 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> BoundType::MetricType **mlpack::tree::BinarySpaceTree**< BoundType, StatisticType, MatType >::Metric ( ) const [inline]

Get the metric which the tree uses.

Definition at line 294 of file binary_space_tree.hpp.

References mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::bound.

21.105.4.31 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> double **mlpack::tree::BinarySpaceTree**< BoundType, StatisticType, MatType >::MinDistance (const **BinarySpaceTree**< BoundType, StatisticType, MatType > * *other*) const [inline]

Return the minimum distance to another node.

Definition at line 355 of file binary_space_tree.hpp.

References mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::bound, and mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::Bound().

21.105.4.32 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> double **mlpack::tree::BinarySpaceTree**< BoundType, StatisticType, MatType >::MinDistance (const arma::vec & *point*) const [inline]

Return the minimum distance to another point.

Definition at line 373 of file binary_space_tree.hpp.

References mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::bound.

21.105.4.33 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> size_t **mlpack::tree::BinarySpaceTree**< BoundType, StatisticType, MatType >::NumChildren ( ) const

Return the number of children in this node.

21.105.4.34 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> size_t mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::NumDescendants ( ) const

Return the number of descendants of this node.

For a non-leaf in a binary space tree, this is the number of points at the descendant leaves. For a leaf, this is the number of points in the leaf.

21.105.4.35 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> size_t mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::NumPoints ( ) const

Return the number of points in this node (0 if not a leaf).

21.105.4.36 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> BinarySpaceTree* mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::Parent ( ) const [inline]

Gets the parent of this node.

Definition at line 279 of file binary_space_tree.hpp.

References mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::parent.

21.105.4.37 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> BinarySpaceTree*& mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::Parent ( ) [inline]

Modify the parent of this node.

Definition at line 281 of file binary_space_tree.hpp.

References mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::parent.

21.105.4.38 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> size_t mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::Point (const size_t *index*) const

Return the index (with reference to the dataset) of a particular point in this node.

This will happily return invalid indices if the given index is greater than the number of points in this node (obtained with **NumPoints()** (p. 473)) – be careful.

Parameters

index	Index of point for which a dataset index is wanted.

21.105.4.39 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> math::Range mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::RangeDistance (const BinarySpaceTree< BoundType, StatisticType, MatType > * *other*) const [inline]

Return the minimum and maximum distance to another node.

Definition at line 367 of file binary_space_tree.hpp.

References mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::bound, and mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::Bound().

21.105.4.40 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> math::Range mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::RangeDistance (const arma::vec & *point*) const [inline]

Return the minimum and maximum distance to another point.

Definition at line 385 of file binary_space_tree.hpp.

References mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::bound.

21.105.4.41 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> BinarySpaceTree* mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::Right ( ) const [inline]

Gets the right child of this node.

Definition at line 274 of file binary_space_tree.hpp.

References mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::right.

21.105.4.42 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> BinarySpaceTree*& mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::Right ( ) [inline]

Modify the right child of this node.

Definition at line 276 of file binary_space_tree.hpp.

References mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::right.

21.105.4.43 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> size_t mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::SplitDimension ( ) const [inline]

Get the split dimension for this node.

Definition at line 284 of file binary_space_tree.hpp.

References mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::splitDimension.

21.105.4.44 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> size_t& mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::SplitDimension ( ) [inline]

Modify the split dimension for this node.

Definition at line 286 of file binary_space_tree.hpp.

References mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::splitDimension.

21.105.4.45 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> void mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::SplitNode (MatType & *data*) [private]

Splits the current node, assigning its left and right children recursively.

Parameters

data	Dataset which we are using.

21.105.4.46 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> void mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::SplitNode (MatType & *data*, std::vector< size_t > & *oldFromNew*) [private]

Splits the current node, assigning its left and right children recursively.

Also returns a list of the changed indices.

Parameters

data	Dataset which we are using.
oldFromNew	Vector holding permuted indices.

21.105.4.47 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> const StatisticType& mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::Stat ( ) const [inline]

Return the statistic object for this node.

Definition at line 253 of file binary_space_tree.hpp.

References mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::stat.

21.105.4.48 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> StatisticType& mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::Stat ( ) [inline]

Return the statistic object for this node.

Definition at line 255 of file binary_space_tree.hpp.

References mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::stat.

21.105.4.49 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> std::string mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::ToString ( ) const

Returns a string representation of this object.

21.105.4.50 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> size_t mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::TreeDepth ( ) const

Obtains the number of levels below this node in the tree, starting with this.

21.105.4.51 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> size_t mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::TreeSize ( ) const

Obtains the number of nodes in the tree, starting with this.

21.105.5 Member Data Documentation

21.105.5.1 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> size_t mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::begin [private]

The index of the first point in the dataset contained in this node (and its children).

Definition at line 63 of file binary_space_tree.hpp.

Referenced by mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::Begin(), and mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::CopyMe().

21.105.5.2 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> BoundType mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::bound [private]

The bound object for this node.

Definition at line 70 of file binary_space_tree.hpp.

Referenced by mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::Bound(), mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::Centroid(), mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::CopyMe(), mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::MaxDistance(), mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::Metric(), mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::MinDistance(), and mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::RangeDistance().

21.105.5.3 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> size_t mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::count [private]

The number of points of the dataset contained in this node (and its children).

Definition at line 66 of file binary_space_tree.hpp.

Referenced by mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::CopyMe(), and mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::Count().

21.105.5.4 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> MatType& mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::dataset [private]

The dataset.

Definition at line 78 of file binary_space_tree.hpp.

Referenced by mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::Dataset().

21.105.5.5 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> double mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::furthestDescendantDistance [private]

The distance to the furthest descendant, cached to speed things up.

Definition at line 76 of file binary_space_tree.hpp.

21.105.5.6 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> size_t mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::leafSize [private]

The leaf size.

Definition at line 68 of file binary_space_tree.hpp.

Referenced by mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::CopyMe(), and mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::LeafSize().

21.105.5.7 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> BinarySpaceTree* mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::left [private]

The left child node.

Definition at line 56 of file binary_space_tree.hpp.

Referenced by mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::Left().

21.105.5.8 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> BinarySpaceTree* mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::parent [private]

The parent node (NULL if this is the root of the tree).

Definition at line 60 of file binary_space_tree.hpp.

Referenced by mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::Parent().

21.105.5.9 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> BinarySpaceTree* mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::right [private]

The right child node.

Definition at line 58 of file binary_space_tree.hpp.

Referenced by mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::Right().

21.105.5.10 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> size_t mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::splitDimension [private]

The dimension this node split on if it is a parent.

Definition at line 74 of file binary_space_tree.hpp.

Referenced by mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::SplitDimension().

21.105.5.11 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> StatisticType mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::stat [private]

Any extra data contained in the node.

Definition at line 72 of file binary_space_tree.hpp.

Referenced by mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::CopyMe(), and mlpack::tree::-BinarySpaceTree< BoundType, StatisticType, MatType >::Stat().

The documentation for this class was generated from the following file:

- src/mlpack/core/tree/binary_space_tree/**binary_space_tree.hpp**

21.106 mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::DualTree-Traverser< RuleType > Class Template Reference

A dual-tree traverser for binary space trees; see dual_tree_traverser.hpp.

Public Member Functions

- **DualTreeTraverser** (RuleType &**rule**)

  Instantiate the dual-tree traverser with the given rule set.
- size_t **NumBaseCases** () const

  Get the number of times a base case was calculated.
- size_t & **NumBaseCases** ()

  Modify the number of times a base case was calculated.
- size_t **NumPrunes** () const

  Get the number of prunes.
- size_t & **NumPrunes** ()

  Modify the number of prunes.
- size_t **NumScores** () const

  Get the number of times a node combination was scored.
- size_t & **NumScores** ()

  Modify the number of times a node combination was scored.
- size_t **NumVisited** () const

  Get the number of visited combinations.
- size_t & **NumVisited** ()

  Modify the number of visited combinations.
- void **Traverse** (**BinarySpaceTree** &queryNode, **BinarySpaceTree** &referenceNode)

  Traverse the two trees.

Private Attributes

- size_t **numBaseCases**

  The number of times a base case was calculated.
- size_t **numPrunes**

  The number of prunes.
- size_t **numScores**

  The number of times a node combination was scored.
- size_t **numVisited**

  The number of node combinations that have been visited during traversal.
- RuleType & **rule**

  Reference to the rules with which the trees will be traversed.

21.106.1 Detailed Description

template<typename BoundType, typename StatisticType = EmptyStatistic, typename MatType = arma::mat>template<typename RuleType>class mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::DualTreeTraverser< RuleType >

A dual-tree traverser for binary space trees; see dual_tree_traverser.hpp.

Definition at line 91 of file binary_space_tree.hpp.

21.106.2 Constructor & Destructor Documentation

21.106.2.1 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> template<typename RuleType > mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::DualTreeTraverser< RuleType >::DualTreeTraverser (RuleType & *rule*)

Instantiate the dual-tree traverser with the given rule set.

21.106.3 Member Function Documentation

21.106.3.1 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> template<typename RuleType > size_t mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::DualTreeTraverser< RuleType >::NumBaseCases ( ) const [inline]

Get the number of times a base case was calculated.

Definition at line 69 of file dual_tree_traverser.hpp.

21.106.3.2 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> template<typename RuleType > size_t& mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::DualTreeTraverser< RuleType >::NumBaseCases ( ) [inline]

Modify the number of times a base case was calculated.

Definition at line 71 of file dual_tree_traverser.hpp.

21.106.3.3 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> template<typename RuleType > size_t mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::DualTreeTraverser< RuleType >::NumPrunes ( ) const [inline]

Get the number of prunes.

Definition at line 54 of file dual_tree_traverser.hpp.

21.106.3.4 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> template<typename RuleType > size_t& mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::DualTreeTraverser< RuleType >::NumPrunes ( ) [inline]

Modify the number of prunes.

Definition at line 56 of file dual_tree_traverser.hpp.

21.106.3.5 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> template<typename RuleType > size_t mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::DualTreeTraverser< RuleType >::NumScores ( ) const [inline]

Get the number of times a node combination was scored.

Definition at line 64 of file dual_tree_traverser.hpp.

21.106.3.6 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> template<typename RuleType > size_t& mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::DualTreeTraverser< RuleType >::NumScores ( ) [inline]

Modify the number of times a node combination was scored.

Definition at line 66 of file dual_tree_traverser.hpp.

21.106.3.7 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> template<typename RuleType > size_t mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::DualTreeTraverser< RuleType >::NumVisited ( ) const [inline]

Get the number of visited combinations.

Definition at line 59 of file dual_tree_traverser.hpp.

21.106.3.8 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> template<typename RuleType > size_t& mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::DualTreeTraverser< RuleType >::NumVisited ( ) [inline]

Modify the number of visited combinations.

Definition at line 61 of file dual_tree_traverser.hpp.

21.106.3.9 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> template<typename RuleType > void mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::DualTreeTraverser< RuleType >::Traverse (BinarySpaceTree & *queryNode*, BinarySpaceTree & *referenceNode*)

Traverse the two trees.

This does not reset the number of prunes.

Parameters

queryNode	The query node to be traversed.
referenceNode	The reference node to be traversed.

21.106.4 Member Data Documentation

21.106.4.1 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> template<typename RuleType > size_t mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::DualTreeTraverser< RuleType >::numBaseCases [private]

The number of times a base case was calculated.

Definition at line 87 of file dual_tree_traverser.hpp.

21.106.4.2 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> template<typename RuleType > size_t mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::DualTreeTraverser< RuleType >::numPrunes [private]

The number of prunes.

Definition at line 78 of file dual_tree_traverser.hpp.

21.106.4.3 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> template<typename RuleType > size_t mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::DualTreeTraverser< RuleType >::numScores [private]

The number of times a node combination was scored.

Definition at line 84 of file dual_tree_traverser.hpp.

21.106.4.4 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> template<typename RuleType > size_t mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::DualTreeTraverser< RuleType >::numVisited [private]

The number of node combinations that have been visited during traversal.

Definition at line 81 of file dual_tree_traverser.hpp.

21.106.4.5 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> template<typename RuleType > RuleType& mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::DualTreeTraverser< RuleType >::rule [private]

Reference to the rules with which the trees will be traversed.

Definition at line 75 of file dual_tree_traverser.hpp.

The documentation for this class was generated from the following files:

- src/mlpack/core/tree/binary_space_tree/**binary_space_tree.hpp**
- src/mlpack/core/tree/binary_space_tree/**dual_tree_traverser.hpp**

21.107 mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::SingleTreeTraverser< RuleType > Class Template Reference

A single-tree traverser for binary space trees; see single_tree_traverser.hpp for implementation.

Public Member Functions

- **SingleTreeTraverser** (RuleType &**rule**)

  Instantiate the single tree traverser with the given rule set.
- size_t **NumPrunes** () const

  Get the number of prunes.

- size_t & **NumPrunes** ()

  Modify the number of prunes.

- void **Traverse** (const size_t queryIndex, **BinarySpaceTree** &referenceNode)

  Traverse the tree with the given point.

Private Attributes

- size_t **numPrunes**

  The number of nodes which have been pruned during traversal.

- RuleType & **rule**

  Reference to the rules with which the tree will be traversed.

21.107.1 Detailed Description

template<typename BoundType, typename StatisticType = EmptyStatistic, typename MatType = arma::mat>template<typename RuleType>class mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::SingleTreeTraverser< RuleType >

A single-tree traverser for binary space trees; see single_tree_traverser.hpp for implementation.

Definition at line 87 of file binary_space_tree.hpp.

21.107.2 Constructor & Destructor Documentation

21.107.2.1 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> template<typename RuleType > mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::SingleTreeTraverser< RuleType >::SingleTreeTraverser (RuleType & *rule*)

Instantiate the single tree traverser with the given rule set.

21.107.3 Member Function Documentation

21.107.3.1 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> template<typename RuleType > size_t mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::SingleTreeTraverser< RuleType >::NumPrunes ( ) const [inline]

Get the number of prunes.

Definition at line 54 of file single_tree_traverser.hpp.

21.107.3.2 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> template<typename RuleType > size_t& mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::SingleTreeTraverser< RuleType >::NumPrunes ( ) [inline]

Modify the number of prunes.

Definition at line 56 of file single_tree_traverser.hpp.

21.107.3.3 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> template<typename RuleType > void mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::SingleTreeTraverser< RuleType >::Traverse (const size_t *queryIndex,* BinarySpaceTree & *referenceNode*)

Traverse the tree with the given point.

Parameters

queryIndex	The index of the point in the query set which is being used as the query point.
referenceNode	The tree node to be traversed.

21.107.4 Member Data Documentation

21.107.4.1 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> template<typename RuleType > size_t mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::SingleTreeTraverser< RuleType >::numPrunes [private]

The number of nodes which have been pruned during traversal.

Definition at line 63 of file single_tree_traverser.hpp.

21.107.4.2 template<typename BoundType , typename StatisticType = EmptyStatistic, typename MatType = arma::mat> template<typename RuleType > RuleType& mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::SingleTreeTraverser< RuleType >::rule [private]

Reference to the rules with which the tree will be traversed.

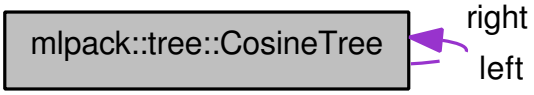
Definition at line 60 of file single_tree_traverser.hpp.

The documentation for this class was generated from the following files:

- src/mlpack/core/tree/binary_space_tree/**binary_space_tree.hpp**
- src/mlpack/core/tree/binary_space_tree/**single_tree_traverser.hpp**

21.108 mlpack::tree::CosineTree Class Reference

Collaboration diagram for mlpack::tree::CosineTree:



Public Member Functions

- **CosineTree** (arma::mat **data**, arma::rowvec **centroid**, arma::vec **probabilities**)

  So other classes can use TreeType::Mat.
- **CosineTree** ()

  Create an empty tree node.
- ~**CosineTree** ()

  Deletes this node, deallocating the memory for the children and calling their destructors in turn.
- arma::rowvec **Centroid** ()

  Returns a reference to the centroid.
- void **Centroid** (arma::rowvec &centr)

  Sets the centroid.

- **CosineTree** & **Child** (const size_t child) const

  Return the specified child (0 will be left, 1 will be right).
- arma::mat **Data** ()

  Returns a reference to the data.
- void **Data** (arma::mat &d)

  Sets a reference to the data.
- **CosineTree** ∗ **Left** () const

  Gets the left child of this node.
- void **Left** (**CosineTree** ∗child)

  Sets the Left child of this node.
- size_t **NumPoints** () const

  Return the number of points in this node (0 if not a leaf).
- arma::vec **Probabilities** ()

  Returns a reference to Sample Probabilites.
- void **Probabilities** (arma::vec &prob)

  Sets a reference to Sample Probabilites.
- **CosineTree** ∗ **Right** () const

  Gets the right child of this node.
- void **Right** (**CosineTree** ∗child)

  Sets the Right child of this node.

Private Attributes

- arma::rowvec **centroid**

  Centroid.
- arma::mat **data**

  Data.
- **CosineTree** ∗ **left**

  The left child node.
- size_t **numPoints**

  Number of points in the node.
- arma::vec **probabilities**

  Sampling Probabilities.
- **CosineTree** ∗ **right**

  The right child node.

21.108.1 Detailed Description

Definition at line 31 of file cosine_tree.hpp.

21.108.2 Constructor & Destructor Documentation

21.108.2.1 mlpack::tree::CosineTree::CosineTree (arma::mat *data,* arma::rowvec *centroid,* arma::vec *probabilities*)

So other classes can use TreeType::Mat.

Constructor

Parameters

data	Dataset to create tree from.
centroid	Centroid of the matrix.
probabilities	Sampling probabilities

21.108.2.2 mlpack::tree::CosineTree::CosineTree ( )

Create an empty tree node.

21.108.2.3 mlpack::tree::CosineTree::~CosineTree ( )

Deletes this node, deallocating the memory for the children and calling their destructors in turn.

This will invalidate any pointers or references to any nodes which are children of this one.

21.108.3 Member Function Documentation

21.108.3.1 arma::rowvec mlpack::tree::CosineTree::Centroid ( )

Returns a reference to the centroid.

21.108.3.2 void mlpack::tree::CosineTree::Centroid (arma::rowvec & *centr*)

Sets the centroid.

21.108.3.3 CosineTree& mlpack::tree::CosineTree::Child (const size_t *child*) const

Return the specified child (0 will be left, 1 will be right).

If the index is greater than 1, this will return the right child.

Parameters

child	Index of child to return.

21.108.3.4 arma::mat mlpack::tree::CosineTree::Data ( )

Returns a reference to the data.

21.108.3.5 void mlpack::tree::CosineTree::Data (arma::mat & *d*)

Sets a reference to the data.

21.108.3.6 CosineTree* mlpack::tree::CosineTree::Left ( ) const

Gets the left child of this node.

21.108.3.7 void mlpack::tree::CosineTree::Left (CosineTree * *child*)

Sets the Left child of this node.

21.108.3.8 size_t mlpack::tree::CosineTree::NumPoints ( ) const

Return the number of points in this node (0 if not a leaf).

21.108.3.9 arma::vec mlpack::tree::CosineTree::Probabilities ( )

Returns a reference to Sample Probabilites.

21.108.3.10 void mlpack::tree::CosineTree::Probabilities (arma::vec & *prob*)

Sets a reference to Sample Probabilites.

21.108.3.11 CosineTree* mlpack::tree::CosineTree::Right ( ) const

Gets the right child of this node.

21.108.3.12 void mlpack::tree::CosineTree::Right (CosineTree * *child*)

Sets the Right child of this node.

21.108.4 Member Data Documentation

21.108.4.1 arma::rowvec mlpack::tree::CosineTree::centroid [private]

Centroid.

Definition at line 37 of file cosine_tree.hpp.

21.108.4.2 arma::mat mlpack::tree::CosineTree::data [private]

Data.

Definition at line 35 of file cosine_tree.hpp.

21.108.4.3 CosineTree* mlpack::tree::CosineTree::left [private]

The left child node.

Definition at line 41 of file cosine_tree.hpp.

21.108.4.4 size_t mlpack::tree::CosineTree::numPoints [private]

Number of points in the node.

Definition at line 45 of file cosine_tree.hpp.

21.108.4.5 arma::vec mlpack::tree::CosineTree::probabilities [private]

Sampling Probabilities.

Definition at line 39 of file cosine_tree.hpp.

21.108.4.6 CosineTree* mlpack::tree::CosineTree::right [private]

The right child node.

Definition at line 43 of file cosine_tree.hpp.

The documentation for this class was generated from the following file:

- src/mlpack/core/tree/cosine_tree/**cosine_tree.hpp**

21.109 mlpack::tree::CosineTreeBuilder Class Reference

Public Member Functions

- **CosineTreeBuilder** ()
  Empty Constructor.
- ~**CosineTreeBuilder** ()
  Destructor.
- void **CTNode** (arma::mat A, **CosineTree** &root)
  Creates a new cosine tree node.
- void **CTNodeSplit** (**CosineTree** &root, **CosineTree** &left, **CosineTree** &right)
  Splits a cosine tree node.

Private Member Functions

- arma::rowvec **CalculateCentroid** (arma::mat A) const
  Calculates the centroid of the matrix.
- void **CreateCosineSimilarityArray** (std::vector< double > &c, arma::mat A, size_t pivot)
  Creates Cosine Similarity Array.
- double **GetMaxSimilarity** (std::vector< double > c)
  Calculates Maximum Cosine Similarity.
- double **GetMinSimilarity** (std::vector< double > c)
  Calculates Maximum Cosine Similarity.
- size_t **GetPivot** (arma::vec prob)
  Calculates the Pivot for splitting.
- void **LSSampling** (arma::mat A, arma::vec &prob)
  Length Square Sampling method for sampling rows of the matrix.
- void **SplitData** (std::vector< double > c, arma::mat &ALeft, arma::mat &Aright, arma::mat A)
  Splits the points into the root node into children nodes.

21.109.1 Detailed Description

Definition at line 33 of file cosine_tree_builder.hpp.

21.109.2 Constructor & Destructor Documentation

21.109.2.1 mlpack::tree::CosineTreeBuilder::CosineTreeBuilder ( )

Empty Constructor.

21.109.2.2 mlpack::tree::CosineTreeBuilder::~CosineTreeBuilder ( )

Destructor.

21.109.3 Member Function Documentation

21.109.3.1 arma::rowvec mlpack::tree::CosineTreeBuilder::CalculateCentroid (arma::mat *A*) const [private]

Calculates the centroid of the matrix.

Parameters

A	Matrix for which the centroid has to be calculated

21.109.3.2 void mlpack::tree::CosineTreeBuilder::CreateCosineSimilarityArray (std::vector< double > & *c*, arma::mat *A*, size_t *pivot*) [private]

Creates Cosine Similarity Array.

Parameters

c	Array of Cosine Similarity
A	All points
pivot	pivot point

21.109.3.3 void mlpack::tree::CosineTreeBuilder::CTNode (arma::mat *A*, CosineTree & *root*)

Creates a new cosine tree node.

Parameters

A	Data for constructing the node
root	Reference to the constructed node

21.109.3.4 void mlpack::tree::CosineTreeBuilder::CTNodeSplit (CosineTree & *root*, CosineTree & *left*, CosineTree & *right*)

Splits a cosine tree node.

Parameters

root	Node to be split

right	reference to the right child
left	reference to the left child

21.109.3.5 double mlpack::tree::CosineTreeBuilder::GetMaxSimilarity (std::vector< double > *c*) [private]

Calculates Maximum Cosine Similarity.

Parameters

c	Array of Cosine Similarities

21.109.3.6 double mlpack::tree::CosineTreeBuilder::GetMinSimilarity (std::vector< double > *c*) [private]

Calculates Maximum Cosine Similarity.

Parameters

c	Array of Cosine Similarities

21.109.3.7 size_t mlpack::tree::CosineTreeBuilder::GetPivot (arma::vec *prob*) [private]

Calculates the Pivot for splitting.

Parameters

prob	Probability for a point to act as the pivot

21.109.3.8 void mlpack::tree::CosineTreeBuilder::LSSampling (arma::mat *A*, arma::vec & *prob*) [private]

Length Square Sampling method for sampling rows of the matrix.

Parameters

A	Matrix for which probabilities are calculated
prob	Reference to the probability vector

21.109.3.9 void mlpack::tree::CosineTreeBuilder::SplitData (std::vector< double > *c*, arma::mat & *ALeft*, arma::mat & *ARight*, arma::mat *A*) [private]

Splits the points into the root node into children nodes.

Parameters

c	Array of Cosin Similarities
ALeft	Matrix for storing the points in Left Node
ARight	Matrix for storing the points in Right Node

A	All points

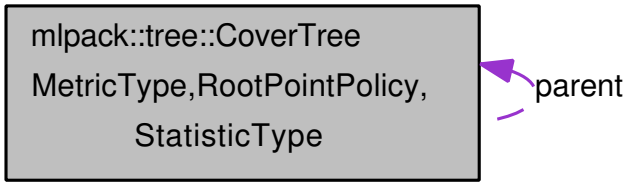
The documentation for this class was generated from the following file:

- src/mlpack/core/tree/cosine_tree/**cosine_tree_builder.hpp**

21.110 mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType > Class Template Reference

A cover tree is a tree specifically designed to speed up nearest-neighbor computation in high-dimensional spaces.

Collaboration diagram for mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >:



Classes

- class **DualTreeTraverser**

  A dual-tree cover tree traverser; see dual_tree_traverser.hpp.
- class **SingleTreeTraverser**

  A single-tree cover tree traverser; see single_tree_traverser.hpp for implementation.

Public Types

- typedef arma::mat **Mat**

Public Member Functions

- **CoverTree** (const arma::mat &**dataset**, const double **base**=2.0, MetricType ∗**metric**=NULL)

  Create the cover tree with the given dataset and given base.
- **CoverTree** (const arma::mat &**dataset**, MetricType &**metric**, const double **base**=2.0)

  Create the cover tree with the given dataset and the given instantiated metric.
- **CoverTree** (const arma::mat &**dataset**, const double **base**, const size_t pointIndex, const int **scale**, **CoverTree** ∗**parent**, const double **parentDistance**, arma::Col< size_t > &indices, arma::vec &distances, size_t nearSetSize, size_t &farSetSize, size_t &usedSetSize, MetricType &**metric**=NULL)

  Construct a child cover tree node.
- **CoverTree** (const arma::mat &**dataset**, const double **base**, const size_t pointIndex, const int **scale**, **CoverTree** ∗**parent**, const double **parentDistance**, const double **furthestDescendantDistance**, MetricType ∗**metric**=NULL)

  *Manually construct a cover tree node; no tree assembly is done in this constructor, and children must be added manually (use **Children()** (p. 499)).*
- **CoverTree** (const **CoverTree** &other)

  Create a cover tree from another tree.

- ~**CoverTree** ()

  Delete this cover tree node and its children.
- double **Base** () const

  Get the base.
- double & **Base** ()

  Modify the base; don't do this, you'll break everything.
- void **Centroid** (arma::vec &centroid) const

  Get the centroid of the node and store it in the given vector.
- const **CoverTree** & **Child** (const size_t index) const

  Get a particular child node.
- **CoverTree** & **Child** (const size_t index)

  Modify a particular child node.
- const std::vector< **CoverTree** * > & **Children** () const

  Get the children.
- std::vector< **CoverTree** * > & **Children** ()

  Modify the children manually (maybe not a great idea).
- const arma::mat & **Dataset** () const

  Get a reference to the dataset.
- size_t **Descendant** (const size_t index) const

  Get the index of a particular descendant point.
- size_t **DistanceComps** () const
- size_t & **DistanceComps** ()
- double **FurthestDescendantDistance** () const

  Get the distance from the center of the node to the furthest descendant.
- double & **FurthestDescendantDistance** ()

  Modify the distance from the center of the node to the furthest descendant.
- double **FurthestPointDistance** () const

  Get the distance to the furthest point. This is always 0 for cover trees.
- bool **IsLeaf** () const
- double **MaxDistance** (const **CoverTree** *other) const

  Return the maximum distance to another node.
- double **MaxDistance** (const **CoverTree** *other, const double distance) const

  Return the maximum distance to another node given that the point-to-point distance has already been calculated.
- double **MaxDistance** (const arma::vec &other) const

  Return the maximum distance to another point.
- double **MaxDistance** (const arma::vec &other, const double distance) const

  Return the maximum distance to another point given that the distance from the center to the point has already been calculated.
- MetricType & **Metric** () const

  Get the instantiated metric.
- double **MinDistance** (const **CoverTree** *other) const

  Return the minimum distance to another node.
- double **MinDistance** (const **CoverTree** *other, const double distance) const

  Return the minimum distance to another node given that the point-to-point distance has already been calculated.
- double **MinDistance** (const arma::vec &other) const

  Return the minimum distance to another point.
- double **MinDistance** (const arma::vec &other, const double distance) const

Return the minimum distance to another point given that the distance from the center to the point has already been calculated.

- size_t **NumChildren** () const

  Get the number of children.

- size_t **NumDescendants** () const

  Get the number of descendant points.

- size_t **NumPoints** () const
- **CoverTree** * **Parent** () const

  Get the parent node.

- **CoverTree** *& **Parent** ()

  Modify the parent node.

- double **ParentDistance** () const

  Get the distance to the parent.

- double & **ParentDistance** ()

  Modify the distance to the parent.

- size_t **Point** () const

  Get the index of the point which this node represents.

- size_t **Point** (const size_t) const

  For compatibility with other trees; the argument is ignored.

- **math::Range RangeDistance** (const **CoverTree** *other) const

  Return the minimum and maximum distance to another node.

- **math::Range RangeDistance** (const **CoverTree** *other, const double distance) const

  Return the minimum and maximum distance to another node given that the point-to-point distance has already been calculated.

- **math::Range RangeDistance** (const arma::vec &other) const

  Return the minimum and maximum distance to another point.

- **math::Range RangeDistance** (const arma::vec &other, const double distance) const

  Return the minimum and maximum distance to another point given that the point-to-point distance has already been calculated.

- int **Scale** () const

  Get the scale of this node.

- int & **Scale** ()

  Modify the scale of this node. Be careful...

- const StatisticType & **Stat** () const

  Get the statistic for this node.

- StatisticType & **Stat** ()

  Modify the statistic for this node.

- std::string **ToString** () const

  Returns a string representation of this object.

Static Public Member Functions

- static bool **HasSelfChildren** ()

  Returns true: this tree does have self-children.

Private Member Functions

- void **ComputeDistances** (const size_t pointIndex, const arma::Col< size_t > &indices, arma::vec &distances, const size_t pointSetSize)

  Fill the vector of distances with the distances between the point specified by pointIndex and each point in the indices array.
- void **CreateChildren** (arma::Col< size_t > &indices, arma::vec &distances, size_t nearSetSize, size_t &farSetSize, size_t &usedSetSize)

  Create the children for this node.
- void **MoveToUsedSet** (arma::Col< size_t > &indices, arma::vec &distances, size_t &nearSetSize, size_t &farSetSize, size_t &usedSetSize, arma::Col< size_t > &childIndices, const size_t childFarSetSize, const size_t childUsedSetSize)
- size_t **PruneFarSet** (arma::Col< size_t > &indices, arma::vec &distances, const double bound, const size_t nearSetSize, const size_t pointSetSize)
- void **RemoveNewImplicitNodes** ()

  Take a look at the last child (the most recently created one) and remove any implicit nodes that have been created.
- size_t **SortPointSet** (arma::Col< size_t > &indices, arma::vec &distances, const size_t childFarSetSize, const size_t childUsedSetSize, const size_t farSetSize)

  Assuming that the list of indices and distances is sorted as [childFarSet | childUsedSet | farSet | usedSet], resort the sets so the organization is [childFarSet | farSet | childUsedSet | usedSet].
- size_t **SplitNearFar** (arma::Col< size_t > &indices, arma::vec &distances, const double bound, const size_t pointSetSize)

  Split the given indices and distances into a near and a far set, returning the number of points in the near set.

Private Attributes

- double **base**

  The base used to construct the tree.
- std::vector< **CoverTree** * > **children**

  The list of children; the first is the self-child.
- const arma::mat & **dataset**

  Reference to the matrix which this tree is built on.
- size_t **distanceComps**
- double **furthestDescendantDistance**

  Distance to the furthest descendant.
- bool **localMetric**

  Whether or not we need to destroy the metric in the destructor.
- MetricType * **metric**

  The metric used for this tree.
- size_t **numDescendants**

  The number of descendant points.
- **CoverTree** * **parent**

  The parent node (NULL if this is the root of the tree).
- double **parentDistance**

  Distance to the parent.
- size_t **point**

  Index of the point in the matrix which this node represents.
- int **scale**

  Scale level of the node.
- StatisticType **stat**

  The instantiated statistic.

21.110.1 Detailed Description

template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic>class mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >

A cover tree is a tree specifically designed to speed up nearest-neighbor computation in high-dimensional spaces.

Each non-leaf node references a point and has a nonzero number of children, including a "self-child" which references the same point. A leaf node represents only one point.

The tree can be thought of as a hierarchy with the root node at the top level and the leaf nodes at the bottom level. Each level in the tree has an assigned 'scale' i. The tree follows these three conditions:

- nesting: the level C_i is a subset of the level C_{i - 1}.
- covering: all node in level C_{i - 1} have at least one node in the level C_i with distance less than or equal to b^i (exactly one of these is a parent of the point in level C_{i - 1}.
- separation: all nodes in level C_i have distance greater than b^i to all other nodes in level C_i.

The value 'b' refers to the base, which is a parameter of the tree. These three properties make the cover tree very good for fast, high-dimensional nearest-neighbor search.

The theoretical structure of the tree contains many 'implicit' nodes which only have a "self-child" (a child referencing the same point, but at a lower scale level). This practical implementation only constructs explicit nodes – non-leaf nodes with more than one child. A leaf node has no children, and its scale level is INT_MIN.

For more information on cover trees, see

```
@inproceedings{
  author = {Beygelzimer, Alina and Kakade, Sham and Langford, John},
  title = {Cover trees for nearest neighbor},
  booktitle = {Proceedings of the 23rd International Conference on Machine
    Learning},
  series = {ICML '06},
  year = {2006},
  pages = {97--104}
}
```

For information on runtime bounds of the nearest-neighbor computation using cover trees, see the following paper, presented at NIPS 2009:

```
@inproceedings{
  author = {Ram, P., and Lee, D., and March, W.B., and Gray, A.G.},
  title = {Linear-time Algorithms for Pairwise Statistical Problems},
  booktitle = {Advances in Neural Information Processing Systems 22},
  editor = {Y. Bengio and D. Schuurmans and J. Lafferty and C.K.I. Williams
    and A. Culotta},
  pages = {1527--1535},
  year = {2009}
}
```

The **CoverTree** (p. 491) class offers three template parameters; a custom metric type can be used with MetricType (this class defaults to the L2-squared metric). The root node's point can be chosen with the RootPointPolicy; by default, the **FirstPointIsRoot** (p. 515) policy is used, meaning the first point in the dataset is used. The StatisticType policy allows you to define statistics which can be gathered during the creation of the tree.

Template Parameters

MetricType	Metric type to use during tree construction.
RootPointPolicy	Determines which point to use as the root node.
StatisticType	Statistic to be used during tree creation.

Definition at line 103 of file cover_tree.hpp.

21.110.2 Member Typedef Documentation

21.110.2.1 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> typedef arma::mat mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::Mat

Definition at line 106 of file cover_tree.hpp.

21.110.3 Constructor & Destructor Documentation

21.110.3.1 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::CoverTree (const arma::mat & *dataset*, const double *base* = 2.0, MetricType * *metric* = NULL)

Create the cover tree with the given dataset and given base.

The dataset will not be modified during the building procedure (unlike **BinarySpaceTree** (p. 459)).

The last argument will be removed in mlpack 1.1.0 (see #274 and #273).

Parameters

dataset	Reference to the dataset to build a tree on.
base	Base to use during tree building (default 2.0).

21.110.3.2 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::CoverTree (const arma::mat & *dataset*, MetricType & *metric*, const double *base* = 2.0)

Create the cover tree with the given dataset and the given instantiated metric.

Optionally, set the base. The dataset will not be modified during the building procedure (unlike **BinarySpaceTree** (p. 459)).

Parameters

dataset	Reference to the dataset to build a tree on.
metric	Instantiated metric to use during tree building.
base	Base to use during tree building (default 2.0).

21.110.3.3 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::CoverTree (const arma::mat & *dataset*, const double *base*, const size_t *pointIndex*, const int *scale*, CoverTree< MetricType, RootPointPolicy, StatisticType > * *parent*, const double *parentDistance*, arma::Col< size_t > & *indices*, arma::vec & *distances*, size_t *nearSetSize*, size_t & *farSetSize*, size_t & *usedSetSize*, MetricType & *metric* = NULL)

Construct a child cover tree node.

This constructor is not meant to be used externally, but it could be used to insert another node into a tree. This procedure uses only one vector for the near set, the far set, and the used set (this is to prevent unnecessary memory allocation in recursive calls to this constructor). Therefore, the size of the near set, far set, and used set must be passed in. The near set will be entirely used up, and some of the far set may be used. The value of usedSetSize will be set to the number of points used in the construction of this node, and the value of farSetSize will be modified to reflect the number of points in the far set *after* the construction of this node.

If you are calling this manually, be careful that the given scale is as small as possible, or you may be creating an implicit node in your tree.

Parameters

dataset	Reference to the dataset to build a tree on.
base	Base to use during tree building.
pointIndex	Index of the point this node references.
scale	Scale of this level in the tree.
parent	Parent of this node (NULL indicates no parent).
parentDistance	Distance to the parent node.
indices	Array of indices, ordered [nearSet | farSet | usedSet]; will be modified to [farSet | usedSet].
distances	Array of distances, ordered the same way as the indices. These represent the distances between the point specified by pointIndex and each point in the indices array.
nearSetSize	Size of the near set; if 0, this will be a leaf.
farSetSize	Size of the far set; may be modified (if this node uses any points in the far set).
usedSetSize	The number of points used will be added to this number.

21.110.3.4 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::CoverTree (const arma::mat & *dataset*, const double *base*, const size_t *pointIndex*, const int *scale*, CoverTree< MetricType, RootPointPolicy, StatisticType > * *parent*, const double *parentDistance*, const double *furthestDescendantDistance*, MetricType * *metric* = NULL)

Manually construct a cover tree node; no tree assembly is done in this constructor, and children must be added manually (use **Children()** (p. 499)).

This constructor is useful when the tree is being "imported" into the **CoverTree** (p. 491) class after being created in some other manner.

Parameters

dataset	Reference to the dataset this node is a part of.
base	Base that was used for tree building.
pointIndex	Index of the point in the dataset which this node refers to.
scale	Scale of this node's level in the tree.

parent	Parent node (NULL indicates no parent).
parentDistance	Distance to parent node point.
furthest-Descendant-Distance	Distance to furthest descendant point.
metric	Instantiated metric (optional).

21.110.3.5 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::CoverTree (const CoverTree< MetricType, RootPointPolicy, StatisticType > & *other*)

Create a cover tree from another tree.

Be careful! This may use a lot of memory and take a lot of time.

Parameters

other	Cover tree to copy from.

21.110.3.6 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::~CoverTree ( )

Delete this cover tree node and its children.

21.110.4 Member Function Documentation

21.110.4.1 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> double mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::Base ( ) const [inline]

Get the base.

Definition at line 262 of file cover_tree.hpp.

References mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::base.

21.110.4.2 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> double& mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::Base ( ) [inline]

Modify the base; don't do this, you'll break everything.

Definition at line 264 of file cover_tree.hpp.

References mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::base.

21.110.4.3 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> void mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::Centroid (arma::vec & *centroid*) const [inline]

Get the centroid of the node and store it in the given vector.

Definition at line 339 of file cover_tree.hpp.

References mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::dataset, and mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::point.

21.110.4.4 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> const CoverTree& mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::Child (const size_t *index*) const [inline]

Get a particular child node.

Definition at line 238 of file cover_tree.hpp.

References mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::children.

21.110.4.5 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> CoverTree& mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::Child (const size_t *index*) [inline]

Modify a particular child node.

Definition at line 240 of file cover_tree.hpp.

References mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::children.

21.110.4.6 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> const std::vector<CoverTree*>& mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::Children ( ) const [inline]

Get the children.

Definition at line 246 of file cover_tree.hpp.

References mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::children.

21.110.4.7 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> std::vector<CoverTree*>& mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::Children ( ) [inline]

Modify the children manually (maybe not a great idea).

Definition at line 248 of file cover_tree.hpp.

References mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::children.

21.110.4.8 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> void mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::ComputeDistances (const size_t *pointIndex,* const arma::Col< size_t > & *indices,* arma::vec & *distances,* const size_t *pointSetSize*) [private]

Fill the vector of distances with the distances between the point specified by pointIndex and each point in the indices array.

The distances of the first pointSetSize points in indices are calculated (so, this does not necessarily need to use all of the points in the arrays).

Parameters

pointIndex	Point to build the distances for.
indices	List of indices to compute distances for.
distances	Vector to store calculated distances in.
pointSetSize	Number of points in arrays to calculate distances for.

21.110.4.9 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> void mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::CreateChildren (arma::Col< size_t > & *indices,* arma::vec & *distances,* size_t *nearSetSize,* size_t & *farSetSize,* size_t & *usedSetSize*) [private]

Create the children for this node.

21.110.4.10 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> const arma::mat& mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::Dataset ( ) const [inline]

Get a reference to the dataset.

Definition at line 227 of file cover_tree.hpp.

References mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::dataset.

21.110.4.11 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> size_t mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::Descendant (const size_t *index*) const

Get the index of a particular descendant point.

21.110.4.12 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> size_t mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::DistanceComps ( ) const [inline]

Definition at line 475 of file cover_tree.hpp.

References mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::distanceComps.

21.110.4.13 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> size_t& mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::DistanceComps ( ) [inline]

Definition at line 476 of file cover_tree.hpp.

References mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::distanceComps.

21.110.4.14 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> double mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::FurthestDescendantDistance ( ) const [inline]

Get the distance from the center of the node to the furthest descendant.

Definition at line 332 of file cover_tree.hpp.

References mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::furthestDescendantDistance.

21.110.4.15 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> double& mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::FurthestDescendantDistance ( ) [inline]

Modify the distance from the center of the node to the furthest descendant.

Definition at line 336 of file cover_tree.hpp.

References mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::furthestDescendantDistance.

21.110.4.16 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> double mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::FurthestPointDistance ( ) const [inline]

Get the distance to the furthest point. This is always 0 for cover trees.

Definition at line 329 of file cover_tree.hpp.

21.110.4.17 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> static bool mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::HasSelfChildren ( ) [inline], [static]

Returns true: this tree does have self-children.

Definition at line 316 of file cover_tree.hpp.

21.110.4.18 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> bool mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::IsLeaf ( ) const [inline]

Definition at line 234 of file cover_tree.hpp.

References mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::children.

21.110.4.19 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> double mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::MaxDistance (const CoverTree< MetricType, RootPointPolicy, StatisticType > * *other*) const

Return the maximum distance to another node.

21.110.4.20 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> double mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::MaxDistance (const CoverTree< MetricType, RootPointPolicy, StatisticType > * *other,* const double *distance*) const

Return the maximum distance to another node given that the point-to-point distance has already been calculated.

21.110.4.21 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> double mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::MaxDistance (const arma::vec & *other*) const

Return the maximum distance to another point.

21.110.4.22 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> double mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::MaxDistance (const arma::vec & *other,* const double *distance*) const

Return the maximum distance to another point given that the distance from the center to the point has already been calculated.

21.110.4.23 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> MetricType& mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::Metric ( ) const [inline]

Get the instantiated metric.

Definition at line 342 of file cover_tree.hpp.

References mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::metric.

21.110.4.24 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> double mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::MinDistance (const CoverTree< MetricType, RootPointPolicy, StatisticType > * *other*) const

Return the minimum distance to another node.

21.110.4.25 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> double mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::MinDistance (const CoverTree< MetricType, RootPointPolicy, StatisticType > * *other,* const double *distance*) const

Return the minimum distance to another node given that the point-to-point distance has already been calculated.

21.110.4.26 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> double mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::MinDistance (const arma::vec & *other*) const

Return the minimum distance to another point.

21.110.4.27 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> double mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::MinDistance (const arma::vec & *other,* const double *distance*) const

Return the minimum distance to another point given that the distance from the center to the point has already been calculated.

21.110.4.28 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> void mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::MoveToUsedSet (arma::Col< size_t > & *indices,* arma::vec & *distances,* size_t & *nearSetSize,* size_t & *farSetSize,* size_t & *usedSetSize,* arma::Col< size_t > & *childIndices,* const size_t *childFarSetSize,* const size_t *childUsedSetSize*) [private]

21.110.4.29 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> size_t mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::NumChildren ( ) const [inline]

Get the number of children.

Definition at line 243 of file cover_tree.hpp.

References mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::children.

21.110.4.30 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> size_t mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::NumDescendants ( ) const

Get the number of descendant points.

21.110.4.31 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> size_t mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::NumPoints ( ) const [inline]

Definition at line 235 of file cover_tree.hpp.

21.110.4.32 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> CoverTree* mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::Parent ( ) const [inline]

Get the parent node.

Definition at line 319 of file cover_tree.hpp.

References mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::parent.

21.110.4.33 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> CoverTree*& mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::Parent ( ) [inline]

Modify the parent node.

Definition at line 321 of file cover_tree.hpp.

References mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::parent.

21.110.4.34 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> double mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::ParentDistance ( ) const [inline]

Get the distance to the parent.

Definition at line 324 of file cover_tree.hpp.

References mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::parentDistance.

21.110.4.35 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> double& mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::ParentDistance ( ) [inline]

Modify the distance to the parent.

Definition at line 326 of file cover_tree.hpp.

References mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::parentDistance.

21.110.4.36 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> size_t mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::Point ( ) const [inline]

Get the index of the point which this node represents.

Definition at line 230 of file cover_tree.hpp.

References mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::point.

21.110.4.37 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> size_t mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::Point (const size_t) const [inline]

For compatibility with other trees; the argument is ignored.

Definition at line 232 of file cover_tree.hpp.

References mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::point.

21.110.4.38 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> size_t mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::PruneFarSet (arma::Col< size_t > & *indices,* arma::vec & *distances,* const double *bound,* const size_t *nearSetSize,* const size_t *pointSetSize*) [private]

21.110.4.39 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> math::Range mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::RangeDistance (const CoverTree< MetricType, RootPointPolicy, StatisticType > * *other*) const

Return the minimum and maximum distance to another node.

21.110.4.40 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> math::Range mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::RangeDistance (const CoverTree< MetricType, RootPointPolicy, StatisticType > * *other,* const double *distance*) const

Return the minimum and maximum distance to another node given that the point-to-point distance has already been calculated.

21.110.4.41 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> math::Range mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::RangeDistance (const arma::vec & *other*) const

Return the minimum and maximum distance to another point.

21.110.4.42 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> math::Range mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::RangeDistance (const arma::vec & *other,* const double *distance*) const

Return the minimum and maximum distance to another point given that the point-to-point distance has already been calculated.

21.110.4.43 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> void mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::RemoveNewImplicitNodes ( ) [private]

Take a look at the last child (the most recently created one) and remove any implicit nodes that have been created.

21.110.4.44 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> int mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::Scale ( ) const [inline]

Get the scale of this node.

Definition at line 257 of file cover_tree.hpp.

References mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::scale.

21.110.4.45 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> int& mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::Scale ( ) [inline]

Modify the scale of this node. Be careful...

Definition at line 259 of file cover_tree.hpp.

References mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::scale.

21.110.4.46 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> size_t mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::SortPointSet (arma::Col< size_t > & *indices,* arma::vec & *distances,* const size_t *childFarSetSize,* const size_t *childUsedSetSize,* const size_t *farSetSize*) [private]

Assuming that the list of indices and distances is sorted as [childFarSet | childUsedSet | farSet | usedSet], resort the sets so the organization is [childFarSet | farSet | childUsedSet | usedSet].

The size_t parameters specify the sizes of each set in the array. Only the ordering of the indices and distances arrays will be modified (not their actual contents).

The size of any of the four sets can be zero and this method will handle that case accordingly.

Parameters

indices	List of indices to sort.
distances	List of distances to sort.
childFarSetSize	Number of points in child far set (childFarSet).
childUsedSetSize	Number of points in child used set (childUsedSet).
farSetSize	Number of points in far set (farSet).

21.110.4.47 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> size_t mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::SplitNearFar (arma::Col< size_t > & *indices*, arma::vec & *distances*, const double *bound*, const size_t *pointSetSize*) [private]

Split the given indices and distances into a near and a far set, returning the number of points in the near set.

The distances must already be initialized. This will order the indices and distances such that the points in the near set make up the first part of the array and the far set makes up the rest: [nearSet | farSet].

Parameters

indices	List of indices; will be reordered.
distances	List of distances; will be reordered.
bound	If the distance is less than or equal to this bound, the point is placed into the near set.
pointSetSize	Size of point set (because we may be sorting a smaller list than the indices vector will hold).

21.110.4.48 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> const StatisticType& mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::Stat ( ) const [inline]

Get the statistic for this node.

Definition at line 267 of file cover_tree.hpp.

References mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::stat.

21.110.4.49 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> StatisticType& mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::Stat ( ) [inline]

Modify the statistic for this node.

Definition at line 269 of file cover_tree.hpp.

References mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::stat.

21.110.4.50 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> std::string mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::ToString ( ) const

Returns a string representation of this object.

21.110.5 Member Data Documentation

21.110.5.1 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> double mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::base [private]

The base used to construct the tree.

Definition at line 358 of file cover_tree.hpp.

Referenced by mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::Base().

21.110.5.2 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> std::vector<CoverTree*> mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::children [private]

The list of children; the first is the self-child.

Definition at line 352 of file cover_tree.hpp.

Referenced by mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::Child(), mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::Children(), mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::IsLeaf(), and mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::NumChildren().

21.110.5.3 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> const arma::mat& mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::dataset [private]

Reference to the matrix which this tree is built on.

Definition at line 346 of file cover_tree.hpp.

Referenced by mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::Centroid(), and mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::Dataset().

21.110.5.4 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> size_t mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::distanceComps [private]

Definition at line 479 of file cover_tree.hpp.

Referenced by mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::DistanceComps().

21.110.5.5 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> double mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::furthestDescendantDistance [private]

Distance to the furthest descendant.

Definition at line 373 of file cover_tree.hpp.

Referenced by mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::FurthestDescendantDistance().

21.110.5.6 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> bool mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::localMetric [private]

Whether or not we need to destroy the metric in the destructor.

Definition at line 376 of file cover_tree.hpp.

21.110.5.7 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> MetricType* mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::metric [private]

The metric used for this tree.

Definition at line 379 of file cover_tree.hpp.

Referenced by mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::Metric().

21.110.5.8 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> size_t mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::numDescendants [private]

The number of descendant points.

Definition at line 364 of file cover_tree.hpp.

21.110.5.9 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> CoverTree* mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::parent [private]

The parent node (NULL if this is the root of the tree).

Definition at line 367 of file cover_tree.hpp.

Referenced by mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::Parent().

21.110.5.10 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> double mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::parentDistance [private]

Distance to the parent.

Definition at line 370 of file cover_tree.hpp.

Referenced by mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::ParentDistance().

21.110.5.11 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> size_t mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::point [private]

Index of the point in the matrix which this node represents.

Definition at line 349 of file cover_tree.hpp.

Referenced by mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::Centroid(), and mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::Point().

21.110.5.12 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> int mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::scale [private]

Scale level of the node.

Definition at line 355 of file cover_tree.hpp.

Referenced by mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::Scale().

21.110.5.13 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> StatisticType mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::stat [private]

The instantiated statistic.

Definition at line 361 of file cover_tree.hpp.

Referenced by mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::Stat().

The documentation for this class was generated from the following file:

- src/mlpack/core/tree/cover_tree/**cover_tree.hpp**

21.111 mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::DualTreeTraverser< RuleType > Class Template Reference

A dual-tree cover tree traverser; see dual_tree_traverser.hpp.

Public Member Functions

- **DualTreeTraverser** (RuleType &**rule**)

  Initialize the dual tree traverser with the given rule type.
- size_t **NumBaseCases** () const
- size_t **NumPrunes** () const

  Get the number of pruned nodes.
- size_t & **NumPrunes** ()

  Modify the number of pruned nodes.
- size_t **NumScores** () const
- size_t **NumVisited** () const
- void **Traverse** (**CoverTree** &queryNode, **CoverTree** &referenceNode)

  Traverse the two specified trees.
- void **Traverse** (**CoverTree** &queryNode, std::map< int, std::vector< **DualCoverTreeMapEntry**< MetricType, RootPointPolicy, StatisticType > > > &referenceMap)

  Helper function for traversal of the two trees.

Private Member Functions

- void **PruneMap** (**CoverTree** &queryNode, std::map< int, std::vector< **DualCoverTreeMapEntry**< MetricType, RootPointPolicy, StatisticType > > > &referenceMap, std::map< int, std::vector< **DualCoverTreeMapEntry**< MetricType, RootPointPolicy, StatisticType > > > &childMap)

Prepare map for recursion.

- void **ReferenceRecursion** (**CoverTree** &queryNode, std::map< int, std::vector< **DualCoverTreeMapEntry**< MetricType, RootPointPolicy, StatisticType > > > &referenceMap)

Private Attributes

- size_t **numPrunes**

  The number of pruned nodes.
- RuleType & **rule**

  The instantiated rule set for pruning branches.

21.111.1 Detailed Description

template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic>template<typename RuleType>class mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::DualTreeTraverser< RuleType >

A dual-tree cover tree traverser; see dual_tree_traverser.hpp.

Definition at line 224 of file cover_tree.hpp.

21.111.2 Constructor & Destructor Documentation

21.111.2.1 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> template<typename RuleType > mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::DualTreeTraverser< RuleType >::DualTreeTraverser (RuleType & *rule*)

Initialize the dual tree traverser with the given rule type.

21.111.3 Member Function Documentation

21.111.3.1 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> template<typename RuleType > size_t mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::DualTreeTraverser< RuleType >::NumBaseCases ( ) const [inline]

Definition at line 70 of file dual_tree_traverser.hpp.

21.111.3.2 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> template<typename RuleType > size_t mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::DualTreeTraverser< RuleType >::NumPrunes ( ) const [inline]

Get the number of pruned nodes.

Definition at line 62 of file dual_tree_traverser.hpp.

21.111.3.3 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> template<typename RuleType > size_t& mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::DualTreeTraverser< RuleType >::NumPrunes ( ) [inline]

Modify the number of pruned nodes.

Definition at line 64 of file dual_tree_traverser.hpp.

21.111.3.4 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> template<typename RuleType > size_t mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::DualTreeTraverser< RuleType >::NumScores ( ) const [inline]

Definition at line 69 of file dual_tree_traverser.hpp.

21.111.3.5 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> template<typename RuleType > size_t mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::DualTreeTraverser< RuleType >::NumVisited ( ) const [inline]

Definition at line 68 of file dual_tree_traverser.hpp.

21.111.3.6 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> template<typename RuleType > void mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::DualTreeTraverser< RuleType >::PruneMap (CoverTree & *queryNode,* std::map< int, std::vector< DualCoverTreeMapEntry< MetricType, RootPointPolicy, StatisticType > > > & *referenceMap,* std::map< int, std::vector< DualCoverTreeMapEntry< MetricType, RootPointPolicy, StatisticType > > > & *childMap*) [private]

Prepare map for recursion.

21.111.3.7 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> template<typename RuleType > void mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::DualTreeTraverser< RuleType >::ReferenceRecursion (CoverTree & *queryNode,* std::map< int, std::vector< DualCoverTreeMapEntry< MetricType, RootPointPolicy, StatisticType > > > & *referenceMap*) [private]

21.111.3.8 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> template<typename RuleType > void mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::DualTreeTraverser< RuleType >::Traverse (CoverTree & *queryNode,* CoverTree & *referenceNode*)

Traverse the two specified trees.

Parameters

queryNode	Root of query tree.
referenceNode	Root of reference tree.

21.111.3.9 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> template<typename RuleType > void mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::DualTreeTraverser< RuleType >::Traverse (CoverTree & *queryNode,* std::map< int, std::vector< DualCoverTreeMapEntry< MetricType, RootPointPolicy, StatisticType > > > & *referenceMap*)

Helper function for traversal of the two trees.

21.111.4 Member Data Documentation

21.111.4.1 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> template<typename RuleType > size_t mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::DualTreeTraverser< RuleType >::numPrunes [private]

The number of pruned nodes.

Definition at line 77 of file dual_tree_traverser.hpp.

21.111.4.2 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> template<typename RuleType > RuleType& mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::DualTreeTraverser< RuleType >::rule [private]

The instantiated rule set for pruning branches.

Definition at line 74 of file dual_tree_traverser.hpp.

The documentation for this class was generated from the following files:

- src/mlpack/core/tree/cover_tree/**cover_tree.hpp**
- src/mlpack/core/tree/cover_tree/**dual_tree_traverser.hpp**

21.112 mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::SingleTreeTraverser< RuleType > Class Template Reference

A single-tree cover tree traverser; see single_tree_traverser.hpp for implementation.

Public Member Functions

- **SingleTreeTraverser** (RuleType &**rule**)

  Initialize the single tree traverser with the given rule.
- size_t **NumPrunes** () const

  Get the number of prunes so far.
- size_t & **NumPrunes** ()

  Set the number of prunes (good for a reset to 0).
- void **Traverse** (const size_t queryIndex, **CoverTree** &referenceNode)

  Traverse the tree with the given point.

Private Attributes

- size_t **numPrunes**

  The number of nodes which have been pruned during traversal.
- RuleType & **rule**

  Reference to the rules with which the tree will be traversed.

21.112.1 Detailed Description

template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic>template<typename RuleType>class mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::SingleTreeTraverser< RuleType >

A single-tree cover tree traverser; see single_tree_traverser.hpp for implementation.

Definition at line 220 of file cover_tree.hpp.

21.112.2 Constructor & Destructor Documentation

21.112.2.1 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> template<typename RuleType > mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::SingleTreeTraverser< RuleType >::SingleTreeTraverser (RuleType & *rule*)

Initialize the single tree traverser with the given rule.

21.112.3 Member Function Documentation

21.112.3.1 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> template<typename RuleType > size_t mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::SingleTreeTraverser< RuleType >::NumPrunes ( ) const [inline]

Get the number of prunes so far.

Definition at line 54 of file single_tree_traverser.hpp.

21.112.3.2 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> template<typename RuleType > size_t& mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::SingleTreeTraverser< RuleType >::NumPrunes ( ) [inline]

Set the number of prunes (good for a reset to 0).

Definition at line 56 of file single_tree_traverser.hpp.

21.112.3.3 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> template<typename RuleType > void mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::SingleTreeTraverser< RuleType >::Traverse (const size_t *queryIndex,* CoverTree & *referenceNode*)

Traverse the tree with the given point.

Parameters

queryIndex	The index of the point in the query set which is used as the query point.
referenceNode	The tree node to be traversed.

21.112.4 Member Data Documentation

21.112.4.1 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> template<typename RuleType > size_t mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::SingleTreeTraverser< RuleType >::numPrunes [private]

The number of nodes which have been pruned during traversal.

Definition at line 63 of file single_tree_traverser.hpp.

21.112.4.2 template<typename MetricType = metric::LMetric<2, true>, typename RootPointPolicy = FirstPointIsRoot, typename StatisticType = EmptyStatistic> template<typename RuleType > RuleType& mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::SingleTreeTraverser< RuleType >::rule [private]

Reference to the rules with which the tree will be traversed.

Definition at line 60 of file single_tree_traverser.hpp.

The documentation for this class was generated from the following files:

- src/mlpack/core/tree/cover_tree/**cover_tree.hpp**
- src/mlpack/core/tree/cover_tree/**single_tree_traverser.hpp**

21.113 mlpack::tree::DualCoverTreeMapEntry< MetricType, RootPointPolicy, StatisticType > Struct Template Reference

Forward declaration of struct to be used for traversal.

21.113.1 Detailed Description

template<typename MetricType, typename RootPointPolicy, typename StatisticType>struct mlpack::tree::DualCoverTreeMapEntry< MetricType, RootPointPolicy, StatisticType >

Forward declaration of struct to be used for traversal.

Definition at line 33 of file dual_tree_traverser.hpp.

The documentation for this struct was generated from the following file:

- src/mlpack/core/tree/cover_tree/**dual_tree_traverser.hpp**

21.114 mlpack::tree::EmptyStatistic Class Reference

Empty statistic if you are not interested in storing statistics in your tree.

Public Member Functions

- **EmptyStatistic** ()
- template<typename TreeType >
  EmptyStatistic (TreeType &)

  This constructor is called when a node is finished being created.
- ~**EmptyStatistic** ()

- std::string **ToString** () const

  Returns a string representation of this object.

21.114.1 Detailed Description

Empty statistic if you are not interested in storing statistics in your tree.

Use this as a template for your own.

Definition at line 34 of file statistic.hpp.

21.114.2 Constructor & Destructor Documentation

21.114.2.1 mlpack::tree::EmptyStatistic::EmptyStatistic ( ) [inline]

Definition at line 37 of file statistic.hpp.

21.114.2.2 mlpack::tree::EmptyStatistic::∼EmptyStatistic ( ) [inline]

Definition at line 38 of file statistic.hpp.

21.114.2.3 template<typename TreeType > mlpack::tree::EmptyStatistic::EmptyStatistic (TreeType &) [inline]

This constructor is called when a node is finished being created.

The node is finished, and its children are finished, but it is not necessarily true that the statistics of other nodes are initialized yet.

Parameters

node	Node which this corresponds to.

Definition at line 48 of file statistic.hpp.

21.114.3 Member Function Documentation

21.114.3.1 std::string mlpack::tree::EmptyStatistic::ToString ( ) const [inline]

Returns a string representation of this object.

Definition at line 54 of file statistic.hpp.

The documentation for this class was generated from the following file:

- src/mlpack/core/tree/**statistic.hpp**

21.115 mlpack::tree::FirstPointIsRoot Class Reference

This class is meant to be used as a choice for the policy class RootPointPolicy of the **CoverTree** (p. 491) class.

Static Public Member Functions

- static size_t **ChooseRoot** (const arma::mat &)

  Return the point to be used as the root point of the cover tree.

21.115.1 Detailed Description

This class is meant to be used as a choice for the policy class RootPointPolicy of the **CoverTree** (p. 491) class.

This policy determines which point is used for the root node of the cover tree. This particular implementation simply chooses the first point in the dataset as the root. A more complex implementation might choose, for instance, the point with least maximum distance to other points (the closest to the "middle").

Definition at line 39 of file first_point_is_root.hpp.

21.115.2 Member Function Documentation

21.115.2.1 static size_t mlpack::tree::FirstPointIsRoot::ChooseRoot (const arma::mat &) [inline], [static]

Return the point to be used as the root point of the cover tree.

This just returns 0.

Definition at line 46 of file first_point_is_root.hpp.

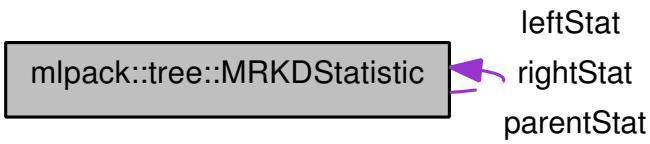
The documentation for this class was generated from the following file:

- src/mlpack/core/tree/cover_tree/**first_point_is_root.hpp**

21.116 mlpack::tree::MRKDStatistic Class Reference

Statistic for multi-resolution kd-trees.

Collaboration diagram for mlpack::tree::MRKDStatistic:



Public Member Functions

- **MRKDStatistic** ()

  Initialize an empty statistic.

- template<typename TreeType >
  MRKDStatistic (const TreeType &)

  This constructor is called when a node is finished initializing.

- size_t **Begin** () const

  Get the index of the initial item in the dataset.

- size_t & **Begin** ()

  Modify the index of the initial item in the dataset.
- const arma::colvec & **CenterOfMass** () const

  Get the center of mass.
- arma::colvec & **CenterOfMass** ()

  Modify the center of mass.
- size_t **Count** () const

  Get the number of items in the dataset.
- size_t & **Count** ()

  Modify the number of items in the dataset.
- size_t **DominatingCentroid** () const

  Get the index of the dominating centroid.
- size_t & **DominatingCentroid** ()

  Modify the index of the dominating centroid.
- std::string **ToString** () const

  Returns a string representation of this object.
- const std::vector< size_t > & **Whitelist** () const

  Access the whitelist.
- std::vector< size_t > & **Whitelist** ()

  Modify the whitelist.

Private Attributes

- size_t **begin**

  The initial item in the dataset, so we don't have to make a copy.
- arma::colvec **centerOfMass**

  The center of mass for this dataset.
- size_t **count**

  The number of items in the dataset.
- const arma::mat * **dataset**

  The data points this object contains.
- size_t **dominatingCentroid**

  The index of the dominating centroid of the associated hyperrectangle.
- bool **isWhitelistValid**

  Whether or not the whitelist is valid.
- const **MRKDStatistic** * **leftStat**

  The left child.
- const **MRKDStatistic** * **parentStat**

  A link to the parent node; NULL if this is the root.
- const **MRKDStatistic** * **rightStat**

  The right child.
- double **sumOfSquaredNorms**

  The sum of the squared Euclidean norms for this dataset.
- std::vector< size_t > **whitelist**

  The list of centroids that cannot own this hyperrectangle.

21.116.1 Detailed Description

Statistic for multi-resolution kd-trees.

Definition at line 33 of file mrkd_statistic.hpp.

21.116.2 Constructor & Destructor Documentation

21.116.2.1 mlpack::tree::MRKDStatistic::MRKDStatistic ( )

Initialize an empty statistic.

21.116.2.2 template<typename TreeType > mlpack::tree::MRKDStatistic::MRKDStatistic (const TreeType &)

This constructor is called when a node is finished initializing.

Parameters

node	The node that has been finished.

21.116.3 Member Function Documentation

21.116.3.1 size_t mlpack::tree::MRKDStatistic::Begin ( ) const [inline]

Get the index of the initial item in the dataset.

Definition at line 53 of file mrkd_statistic.hpp.

References begin.

21.116.3.2 size_t& mlpack::tree::MRKDStatistic::Begin ( ) [inline]

Modify the index of the initial item in the dataset.

Definition at line 55 of file mrkd_statistic.hpp.

References begin.

21.116.3.3 const arma::colvec& mlpack::tree::MRKDStatistic::CenterOfMass ( ) const [inline]

Get the center of mass.

Definition at line 63 of file mrkd_statistic.hpp.

References centerOfMass.

21.116.3.4 arma::colvec& mlpack::tree::MRKDStatistic::CenterOfMass ( ) [inline]

Modify the center of mass.

Definition at line 65 of file mrkd_statistic.hpp.

References centerOfMass.

21.116.3.5 size_t mlpack::tree::MRKDStatistic::Count ( ) const [inline]

Get the number of items in the dataset.

Definition at line 58 of file mrkd_statistic.hpp.

References count.

21.116.3.6 size_t& mlpack::tree::MRKDStatistic::Count ( ) [inline]

Modify the number of items in the dataset.

Definition at line 60 of file mrkd_statistic.hpp.

References count.

21.116.3.7 size_t mlpack::tree::MRKDStatistic::DominatingCentroid ( ) const [inline]

Get the index of the dominating centroid.

Definition at line 68 of file mrkd_statistic.hpp.

References dominatingCentroid.

21.116.3.8 size_t& mlpack::tree::MRKDStatistic::DominatingCentroid ( ) [inline]

Modify the index of the dominating centroid.

Definition at line 70 of file mrkd_statistic.hpp.

References dominatingCentroid.

21.116.3.9 std::string mlpack::tree::MRKDStatistic::ToString ( ) const

Returns a string representation of this object.

21.116.3.10 const std::vector<size_t>& mlpack::tree::MRKDStatistic::Whitelist ( ) const [inline]

Access the whitelist.

Definition at line 73 of file mrkd_statistic.hpp.

References whitelist.

21.116.3.11 std::vector<size_t>& mlpack::tree::MRKDStatistic::Whitelist ( ) [inline]

Modify the whitelist.

Definition at line 75 of file mrkd_statistic.hpp.

References whitelist.

21.116.4 Member Data Documentation

21.116.4.1 size_t mlpack::tree::MRKDStatistic::begin [private]

The initial item in the dataset, so we don't have to make a copy.

Definition at line 81 of file mrkd_statistic.hpp.

Referenced by Begin().

21.116.4.2 arma::colvec mlpack::tree::MRKDStatistic::centerOfMass [private]

The center of mass for this dataset.

Definition at line 93 of file mrkd_statistic.hpp.

Referenced by CenterOfMass().

21.116.4.3 size_t mlpack::tree::MRKDStatistic::count [private]

The number of items in the dataset.

Definition at line 83 of file mrkd_statistic.hpp.

Referenced by Count().

21.116.4.4 const arma::mat* mlpack::tree::MRKDStatistic::dataset [private]

The data points this object contains.

Definition at line 79 of file mrkd_statistic.hpp.

21.116.4.5 size_t mlpack::tree::MRKDStatistic::dominatingCentroid [private]

The index of the dominating centroid of the associated hyperrectangle.

Definition at line 99 of file mrkd_statistic.hpp.

Referenced by DominatingCentroid().

21.116.4.6 bool mlpack::tree::MRKDStatistic::isWhitelistValid [private]

Whether or not the whitelist is valid.

Definition at line 104 of file mrkd_statistic.hpp.

21.116.4.7 const MRKDStatistic* mlpack::tree::MRKDStatistic::leftStat [private]

The left child.

Definition at line 85 of file mrkd_statistic.hpp.

21.116.4.8 const MRKDStatistic* mlpack::tree::MRKDStatistic::parentStat [private]

A link to the parent node; NULL if this is the root.

Definition at line 89 of file mrkd_statistic.hpp.

21.116.4.9 const MRKDStatistic* mlpack::tree::MRKDStatistic::rightStat [private]

The right child.

Definition at line 87 of file mrkd_statistic.hpp.

21.116.4.10 double mlpack::tree::MRKDStatistic::sumOfSquaredNorms [private]

The sum of the squared Euclidean norms for this dataset.

Definition at line 95 of file mrkd_statistic.hpp.

21.116.4.11 std::vector<size_t> mlpack::tree::MRKDStatistic::whitelist [private]

The list of centroids that cannot own this hyperrectangle.

Definition at line 102 of file mrkd_statistic.hpp.

Referenced by Whitelist().

The documentation for this class was generated from the following file:

- src/mlpack/core/tree/**mrkd_statistic.hpp**

21.117 mlpack::tree::TreeTraits< TreeType > Class Template Reference

The **TreeTraits** (p. 521) class provides compile-time information on the characteristics of a given tree type.

Static Public Attributes

- static const bool **FirstPointIsCentroid** = false

  This is true if Point(0) is the centroid of the node.
- static const bool **HasOverlappingChildren** = true

  This is true if the subspaces represented by the children of a node can overlap.
- static const bool **HasParentDistance** = false

  This is true if TreeType::ParentDistance() exists and works.
- static const bool **HasSelfChildren** = false

  This is true if the points contained in the first child of a node (Child(0)) are also contained in that node.

21.117.1 Detailed Description

template<typename TreeType>class mlpack::tree::TreeTraits< TreeType >

The **TreeTraits** (p. 521) class provides compile-time information on the characteristics of a given tree type.

These include traits such as whether or not a node knows the distance to its parent node, or whether or not the subspaces represented by children can overlap.

These traits can be used for static compile-time optimization:

```
// This if statement will be optimized out at compile time!
if (TreeTraits<TreeType>::HasOverlappingChildren == false)
{
  // Do a simpler computation because no children overlap.
}
else
{
  // Do the full, complex calculation.
}
```

The traits can also be used in conjunction with SFINAE to write specialized versions of functions:

```
template<typename TreeType>
void Compute(TreeType& node,
             boost::enable_if<
                 TreeTraits<TreeType>::HasParentDistance>::type*)
{
  // Computation with TreeType::ParentDistance().
}

template<typename TreeType>
void Compute(TreeType& node,
             boost::enable_if<
                 !TreeTraits<TreeType>::HasParentDistance>::type*)
{
  // Computation without TreeType::ParentDistance().
}
```

In those two examples, the boost::enable_if<> class takes a boolean template parameter which allows that function to be called when the boolean is true.

Each trait must be a static const value and not a function; only const values can be used as template parameters (with the exception of constexprs, which are a C++11 feature; but MLPACK is not using C++11). By default (the unspecialized implementation of **TreeTraits** (p. 521)), each parameter is set to make as few assumptions about the tree as possible; so, even if **TreeTraits** (p. 521) is not specialized for a particular tree type, tree-based algorithms should still work.

When you write your own tree, you must specialize the **TreeTraits** (p. 521) class to your tree type and set the corresponding values appropriately. See **mlpack/core/tree/binary_space_tree/traits.hpp** (p. 609) for an example.

Definition at line 87 of file tree_traits.hpp.

21.117.2 Member Data Documentation

21.117.2.1 template<typename TreeType> const bool mlpack::tree::TreeTraits< TreeType >::FirstPointIsCentroid = false [static]

This is true if Point(0) is the centroid of the node.

Definition at line 106 of file tree_traits.hpp.

21.117.2.2 template<typename TreeType> const bool mlpack::tree::TreeTraits< TreeType >::HasOverlappingChildren = true [static]

This is true if the subspaces represented by the children of a node can overlap.

Definition at line 101 of file tree_traits.hpp.

21.117.2.3 template<typename TreeType> const bool mlpack::tree::TreeTraits< TreeType >::HasParentDistance = false [static]

This is true if TreeType::ParentDistance() exists and works.

The ParentDistance() function returns the distance between the center of a node and the center of its parent.

Definition at line 95 of file tree_traits.hpp.

21.117.2.4 template<typename TreeType > const bool mlpack::tree::TreeTraits< TreeType >::HasSelfChildren = false [static]

This is true if the points contained in the first child of a node (Child(0)) are also contained in that node.

Definition at line 112 of file tree_traits.hpp.

The documentation for this class was generated from the following file:

- src/mlpack/core/tree/**tree_traits.hpp**

21.118 mlpack::tree::TreeTraits< BinarySpaceTree< BoundType, StatisticType, MatType > > Class Template Reference

This is a specialization of the TreeType class to the **BinarySpaceTree** (p. 459) tree type.

Static Public Attributes

- static const bool **FirstPointIsCentroid** = false

  There is no guarantee that the first point in a node is its centroid.
- static const bool **HasOverlappingChildren** = false

  Each binary space tree node has two children which represent non-overlapping subsets of the space which the node represents.
- static const bool **HasParentDistance** = false

  The binary space tree cannot easily calculate the distance from a node to its parent; so BinarySpaceTree<...>::ParentDistance() does not exist.
- static const bool **HasSelfChildren** = false

  Points are not contained at multiple levels of the binary space tree.

21.118.1 Detailed Description

template<typename BoundType, typename StatisticType, typename MatType>class mlpack::tree::TreeTraits< BinarySpaceTree< BoundType, StatisticType, MatType > >

This is a specialization of the TreeType class to the **BinarySpaceTree** (p. 459) tree type.

It defines characteristics of the binary space tree, and is used to help write tree-independent (but still optimized) tree-based algorithms. See **mlpack/core/tree/tree_traits.hpp** (p. 622) for more information.

Definition at line 39 of file traits.hpp.

21.118.2 Member Data Documentation

21.118.2.1 template<typename BoundType , typename StatisticType , typename MatType > const bool mlpack::tree::TreeTraits< BinarySpaceTree< BoundType, StatisticType, MatType > >::FirstPointIsCentroid = false [static]

There is no guarantee that the first point in a node is its centroid.

Definition at line 58 of file traits.hpp.

21.118.2.2 template<typename BoundType , typename StatisticType , typename MatType > const bool mlpack::tree::TreeTraits< BinarySpaceTree< BoundType, StatisticType, MatType > >::HasOverlappingChildren = false [static]

Each binary space tree node has two children which represent non-overlapping subsets of the space which the node represents.

Therefore, children are not overlapping.

Definition at line 53 of file traits.hpp.

21.118.2.3 template<typename BoundType , typename StatisticType , typename MatType > const bool mlpack::tree::TreeTraits< BinarySpaceTree< BoundType, StatisticType, MatType > >::HasParentDistance = false [static]

The binary space tree cannot easily calculate the distance from a node to its parent; so BinarySpaceTree<...>::ParentDistance() does not exist.

Definition at line 46 of file traits.hpp.

21.118.2.4 template<typename BoundType , typename StatisticType , typename MatType > const bool mlpack::tree::TreeTraits< BinarySpaceTree< BoundType, StatisticType, MatType > >::HasSelfChildren = false [static]

Points are not contained at multiple levels of the binary space tree.

Definition at line 63 of file traits.hpp.

The documentation for this class was generated from the following file:

- src/mlpack/core/tree/binary_space_tree/**traits.hpp**

21.119 mlpack::tree::TreeTraits< CoverTree< MetricType, RootPointPolicy, StatisticType > > Class Template Reference

The specialization of the **TreeTraits** (p. 521) class for the **CoverTree** (p. 491) tree type.

Static Public Attributes

- static const bool **FirstPointIsCentroid** = true

  Each cover tree node contains only one point, and that point is its centroid.
- static const bool **HasOverlappingChildren** = true

  The cover tree (or, this implementation of it) does not require that children represent non-overlapping subsets of the parent node.

- static const bool **HasParentDistance** = true

  *The cover tree calculates the distance between parent and child during construction, so that value is saved and **CoverTree<...>::ParentDistance()** (p. 504) does exist.*

- static const bool **HasSelfChildren** = true

  Cover trees do have self-children.

21.119.1 Detailed Description

template<typename MetricType, typename RootPointPolicy, typename StatisticType>class mlpack::tree::TreeTraits< CoverTree< MetricType, RootPointPolicy, StatisticType > >

The specialization of the **TreeTraits** (p. 521) class for the **CoverTree** (p. 491) tree type.

It defines characteristics of the cover tree, and is used to help write tree-independent (but still optimized) tree-based algorithms. See **mlpack/core/tree/tree_traits.hpp** (p. 622) for more information.

Definition at line 40 of file traits.hpp.

21.119.2 Member Data Documentation

21.119.2.1 template<typename MetricType , typename RootPointPolicy , typename StatisticType > const bool mlpack::tree::TreeTraits< CoverTree< MetricType, RootPointPolicy, StatisticType > >::FirstPointIsCentroid = true [static]

Each cover tree node contains only one point, and that point is its centroid.

Definition at line 60 of file traits.hpp.

21.119.2.2 template<typename MetricType , typename RootPointPolicy , typename StatisticType > const bool mlpack::tree::TreeTraits< CoverTree< MetricType, RootPointPolicy, StatisticType > >::HasOverlappingChildren = true [static]

The cover tree (or, this implementation of it) does not require that children represent non-overlapping subsets of the parent node.

Definition at line 54 of file traits.hpp.

21.119.2.3 template<typename MetricType , typename RootPointPolicy , typename StatisticType > const bool mlpack::tree::TreeTraits< CoverTree< MetricType, RootPointPolicy, StatisticType > >::HasParentDistance = true [static]

The cover tree calculates the distance between parent and child during construction, so that value is saved and **CoverTree<...>::ParentDistance()** (p. 504) does exist.

Definition at line 48 of file traits.hpp.

21.119.2.4 template<typename MetricType , typename RootPointPolicy , typename StatisticType > const bool mlpack::tree::TreeTraits< CoverTree< MetricType, RootPointPolicy, StatisticType > >::HasSelfChildren = true [static]

Cover trees do have self-children.

Definition at line 65 of file traits.hpp.

The documentation for this class was generated from the following file:

- src/mlpack/core/tree/cover_tree/**traits.hpp**

21.120 mlpack::util::CLIDeleter Class Reference

Extremely simple class whose only job is to delete the existing **CLI** (p. 132) object at the end of execution.

Public Member Functions

- **CLIDeleter** ()
- ∼**CLIDeleter** ()

21.120.1 Detailed Description

Extremely simple class whose only job is to delete the existing **CLI** (p. 132) object at the end of execution.

This is meant to allow the user to avoid typing '**CLI::Destroy()** (p. 141)' at the end of their program. The file also defines a static **CLIDeleter** (p. 526) class, which will be initialized at the beginning of the program and deleted at the end. The destructor destroys the **CLI** (p. 132) singleton.

Definition at line 35 of file cli_deleter.hpp.

21.120.2 Constructor & Destructor Documentation

21.120.2.1 mlpack::util::CLIDeleter::CLIDeleter ( )

21.120.2.2 mlpack::util::CLIDeleter::∼CLIDeleter ( )

The documentation for this class was generated from the following file:

- src/mlpack/core/util/**cli_deleter.hpp**

21.121 mlpack::util::NullOutStream Class Reference

Used for **Log::Debug** (p. 280) when not compiled with debugging symbols.

Public Member Functions

- **NullOutStream** ()

  Does nothing.
- **NullOutStream** (const **NullOutStream** &)

  Does nothing.
- **NullOutStream** & **operator**<< (bool val)

  Does nothing.

- **NullOutStream** & **operator**<< (short val)
  Does nothing.
- **NullOutStream** & **operator**<< (unsigned short val)
  Does nothing.
- **NullOutStream** & **operator**<< (int val)
  Does nothing.
- **NullOutStream** & **operator**<< (unsigned int val)
  Does nothing.
- **NullOutStream** & **operator**<< (long val)
  Does nothing.
- **NullOutStream** & **operator**<< (unsigned long val)
  Does nothing.
- **NullOutStream** & **operator**<< (float val)
  Does nothing.
- **NullOutStream** & **operator**<< (double val)
  Does nothing.
- **NullOutStream** & **operator**<< (long double val)
  Does nothing.
- **NullOutStream** & **operator**<< (void *val)
  Does nothing.
- **NullOutStream** & **operator**<< (const char *str)
  Does nothing.
- **NullOutStream** & **operator**<< (std::string &str)
  Does nothing.
- **NullOutStream** & **operator**<< (std::streambuf *sb)
  Does nothing.
- **NullOutStream** & **operator**<< (std::ostream &(*pf)(std::ostream &))
  Does nothing.
- **NullOutStream** & **operator**<< (std::ios &(*pf)(std::ios &))
  Does nothing.
- **NullOutStream** & **operator**<< (std::ios_base &(*pf)(std::ios_base &))
  Does nothing.
- template<typename T >
  NullOutStream & **operator**<< (T &s)
  Does nothing.

21.121.1 Detailed Description

Used for **Log::Debug** (p. 280) when not compiled with debugging symbols.

This class does nothing and should be optimized out entirely by the compiler.

Definition at line 37 of file nulloutstream.hpp.

21.121.2 Constructor & Destructor Documentation

21.121.2.1 mlpack::util::NullOutStream::NullOutStream ( ) [inline]

Does nothing.

Definition at line 43 of file nulloutstream.hpp.

21.121.2.2 mlpack::util::NullOutStream::NullOutStream (const NullOutStream &) [inline]

Does nothing.

Definition at line 48 of file nulloutstream.hpp.

21.121.3 Member Function Documentation

21.121.3.1 NullOutStream& mlpack::util::NullOutStream::operator<< (bool *val*) [inline]

Does nothing.

Definition at line 58 of file nulloutstream.hpp.

21.121.3.2 NullOutStream& mlpack::util::NullOutStream::operator<< (short *val*) [inline]

Does nothing.

Definition at line 60 of file nulloutstream.hpp.

21.121.3.3 NullOutStream& mlpack::util::NullOutStream::operator<< (unsigned short *val*) [inline]

Does nothing.

Definition at line 62 of file nulloutstream.hpp.

21.121.3.4 NullOutStream& mlpack::util::NullOutStream::operator<< (int *val*) [inline]

Does nothing.

Definition at line 64 of file nulloutstream.hpp.

21.121.3.5 NullOutStream& mlpack::util::NullOutStream::operator<< (unsigned int *val*) [inline]

Does nothing.

Definition at line 66 of file nulloutstream.hpp.

21.121.3.6 NullOutStream& mlpack::util::NullOutStream::operator<< (long *val*) [inline]

Does nothing.

Definition at line 68 of file nulloutstream.hpp.

21.121.3.7 NullOutStream& mlpack::util::NullOutStream::operator<< (unsigned long *val*) [inline]

Does nothing.

Definition at line 70 of file nulloutstream.hpp.

21.121.3.8 NullOutStream& mlpack::util::NullOutStream::operator<< (float *val*) [inline]

Does nothing.

Definition at line 72 of file nulloutstream.hpp.

21.121.3.9 NullOutStream& mlpack::util::NullOutStream::operator<< (double *val*) [inline]

Does nothing.

Definition at line 74 of file nulloutstream.hpp.

21.121.3.10 NullOutStream& mlpack::util::NullOutStream::operator<< (long double *val*) [inline]

Does nothing.

Definition at line 76 of file nulloutstream.hpp.

21.121.3.11 NullOutStream& mlpack::util::NullOutStream::operator<< (void * *val*) [inline]

Does nothing.

Definition at line 78 of file nulloutstream.hpp.

21.121.3.12 NullOutStream& mlpack::util::NullOutStream::operator<< (const char * *str*) [inline]

Does nothing.

Definition at line 80 of file nulloutstream.hpp.

21.121.3.13 NullOutStream& mlpack::util::NullOutStream::operator<< (std::string & *str*) [inline]

Does nothing.

Definition at line 82 of file nulloutstream.hpp.

21.121.3.14 NullOutStream& mlpack::util::NullOutStream::operator<< (std::streambuf * *sb*) [inline]

Does nothing.

Definition at line 84 of file nulloutstream.hpp.

21.121.3.15 NullOutStream& mlpack::util::NullOutStream::operator<< (std::ostream &(*)(std::ostream &) *pf*) [inline]

Does nothing.

Definition at line 86 of file nulloutstream.hpp.

21.121.3.16 NullOutStream& mlpack::util::NullOutStream::operator<< (std::ios &(*)(std::ios &) *pf*) [inline]

Does nothing.

Definition at line 89 of file nulloutstream.hpp.

21.121.3.17 NullOutStream& mlpack::util::NullOutStream::operator<< (std::ios_base &(*)(std::ios_base &) *pf*) [inline]

Does nothing.

Definition at line 91 of file nulloutstream.hpp.

21.121.3.18 template<typename T > NullOutStream& mlpack::util::NullOutStream::operator<< (T & *s*) [inline]

Does nothing.

Definition at line 96 of file nulloutstream.hpp.

The documentation for this class was generated from the following file:

- src/mlpack/core/util/**nulloutstream.hpp**

21.122 mlpack::util::Option< N > Class Template Reference

A static object whose constructor registers a parameter with the **CLI** (p. 132) class.

Public Member Functions

- **Option** (bool ignoreTemplate, N defaultValue, const std::string &identifier, const std::string &description, const std::string &parent=std::string(""), bool required=false)

  *Construct an **Option** (p. 530) object.*
- **Option** (const std::string &identifier, const std::string &description, const std::string &parent=std::string(""))

  *Constructs an **Option** (p. 530) object.*

21.122.1 Detailed Description

template<typename N>class mlpack::util::Option< N >

A static object whose constructor registers a parameter with the **CLI** (p. 132) class.

This should not be used outside of **CLI** (p. 132) itself, and you should use the **PARAM_FLAG()** (p. 626), **PARAM_DOUBLE()** (p. 625), **PARAM_INT()** (p. 627), **PARAM_STRING()** (p. 628), or other similar macros to declare these objects instead of declaring them directly.

See Also

core/io/cli.hpp, **mlpack::CLI** (p. 132)

Definition at line 42 of file option.hpp.

21.122.2 Constructor & Destructor Documentation

21.122.2.1 template<typename N > mlpack::util::Option< N >::Option (bool *ignoreTemplate,* N *defaultValue,* const std::string & *identifier,* const std::string & *description,* const std::string & *parent* = `std::string("")`, bool *required* = `false`)

Construct an **Option** (p. 530) object.

When constructed, it will register itself with **CLI** (p. 132).

Parameters

ignoreTemplate	Whether or not the template type matters for this option. Essentially differs options with no value (flags) from those that do, and thus require a type.
defaultValue	Default value this parameter will be initialized to.
identifier	The name of the option (no dashes in front; for –help, we would pass "help").
description	A short string describing the option.
parent	Full pathname of the parent module that "owns" this option. The default is the root node (an empty string).
required	Whether or not the option is required at runtime.

21.122.2.2 template<typename N > mlpack::util::Option< N >::Option (const std::string & *identifier,* const std::string & *description,* const std::string & *parent* = `std::string("")`)

Constructs an **Option** (p. 530) object.

When constructed, it will register a flag with **CLI** (p. 132).

Parameters

identifier	The name of the option (no dashes in front); for –help we would pass "help".
description	A short string describing the option.
parent	Full pathname of the parent module that "owns" this option. The default is the root node (an empty string).

The documentation for this class was generated from the following file:

- src/mlpack/core/util/**option.hpp**

21.123 mlpack::util::PrefixedOutStream Class Reference

Allows us to output to an ostream with a prefix at the beginning of each line, in the same way we would output to cout or cerr.

Public Member Functions

- **PrefixedOutStream** (std::ostream &**destination**, const char ∗**prefix**, bool **ignoreInput**=false, bool **fatal**=false)

  *Set up the **PrefixedOutStream** (p. 532).*
- **PrefixedOutStream** & **operator**<< (bool val)

  Write a bool to the stream.
- **PrefixedOutStream** & **operator**<< (short val)

  Write a short to the stream.
- **PrefixedOutStream** & **operator**<< (unsigned short val)

  Write an unsigned short to the stream.
- **PrefixedOutStream** & **operator**<< (int val)

  Write an int to the stream.
- **PrefixedOutStream** & **operator**<< (unsigned int val)

  Write an unsigned int to the stream.
- **PrefixedOutStream** & **operator**<< (long val)

  Write a long to the stream.

- **PrefixedOutStream** & **operator**<< (unsigned long val)

  Write an unsigned long to the stream.
- **PrefixedOutStream** & **operator**<< (float val)

  Write a float to the stream.
- **PrefixedOutStream** & **operator**<< (double val)

  Write a double to the stream.
- **PrefixedOutStream** & **operator**<< (long double val)

  Write a long double to the stream.
- **PrefixedOutStream** & **operator**<< (void *val)

  Write a void pointer to the stream.
- **PrefixedOutStream** & **operator**<< (const char *str)

  Write a character array to the stream.
- **PrefixedOutStream** & **operator**<< (std::string &str)

  Write a string to the stream.
- **PrefixedOutStream** & **operator**<< (std::streambuf *sb)

  Write a streambuf to the stream.
- **PrefixedOutStream** & **operator**<< (std::ostream &(*pf)(std::ostream &))

  Write an ostream manipulator function to the stream.
- **PrefixedOutStream** & **operator**<< (std::ios &(*pf)(std::ios &))

  Write an ios manipulator function to the stream.
- **PrefixedOutStream** & **operator**<< (std::ios_base &(*pf)(std::ios_base &))

  Write an ios_base manipulator function to the stream.
- template<typename T >
  PrefixedOutStream & **operator**<< (const T &s)

  Write anything else to the stream.

Public Attributes

- std::ostream & **destination**

  The output stream that all data is to be sent too; example: std::cout.
- bool **ignoreInput**

  Discards input, prints nothing if true.

Private Member Functions

- template<typename T >
  void **BaseLogic** (const T &val)

  Conducts the base logic required in all the operator << overloads.
- template<typename T >
  void **CallBaseLogic** (const T &s, typename boost::disable_if< boost::is_class< T > >::type *=0)

  This handles forwarding all primitive types transparently.
- template<typename T >
  void **CallBaseLogic** (const T &s, typename boost::enable_if< boost::is_class< T > >::type *=0, typename boost::disable_if< HasToString< T, std::string(T::*)() const > >::type *=0)

  Forward all objects that do not implement a ToString() method.
- template<typename T >
  void **CallBaseLogic** (const T &s, typename boost::enable_if< boost::is_class< T > >::type *=0, typename boost::enable_if< HasToString< T, std::string(T::*)() const > >::type *=0)

Call ToString() on all objects that implement ToString() before forwarding.

- void **PrefixIfNeeded** ()

  Output the prefix, but only if we need to and if we are allowed to.

Private Attributes

- bool **carriageReturned**

  If true, the previous call to operator<< encountered a CR, and a prefix will be necessary.

- bool **fatal**

  If true, the application will terminate with an error code when a CR is encountered.

- std::string **prefix**

  Contains the prefix we must prepend to each line.

21.123.1 Detailed Description

Allows us to output to an ostream with a prefix at the beginning of each line, in the same way we would output to cout or cerr.

The prefix is specified in the constructor (as well as the destination ostream). A newline must be passed to the stream, and then the prefix will be prepended to the next line. For example,

```
PrefixedOutStream outstr(std::cout, "[TEST] ");
outstr << "Hello world I like " << 7.5;
outstr << "...Continue" << std::endl;
outstr << "After the CR\n" << std::endl;
```

would give, on std::cout,

```
[TEST] Hello world I like 7.5...Continue
[TEST] After the CR
[TEST]
```

These objects are used for the **mlpack::Log** (p. 279) levels (DEBUG, INFO, WARN, and FATAL).

Definition at line 66 of file prefixedoutstream.hpp.

21.123.2 Constructor & Destructor Documentation

21.123.2.1 mlpack::util::PrefixedOutStream::PrefixedOutStream (std::ostream & *destination*, const char * *prefix*, bool *ignoreInput* = `false`, bool *fatal* = `false`) [inline]

Set up the **PrefixedOutStream** (p. 532).

Parameters

destination	ostream which receives output from this object.
prefix	The prefix to prepend to each line.

Definition at line 75 of file prefixedoutstream.hpp.

21.123.3 Member Function Documentation

21.123.3.1 template<typename T > void mlpack::util::PrefixedOutStream::BaseLogic (const T & *val*) [private]

Conducts the base logic required in all the operator << overloads.

Mostly just a good idea to reduce copy-pasta.

Template Parameters

T	The type of the data to output.

Parameters

val	The The data to be output.

21.123.3.2 template<typename T > void mlpack::util::PrefixedOutStream::CallBaseLogic (const T & *s*, typename boost::disable_if< boost::is_class< T > >::type * = 0) [private]

This handles forwarding all primitive types transparently.

21.123.3.3 template<typename T > void mlpack::util::PrefixedOutStream::CallBaseLogic (const T & *s*, typename boost::enable_if< boost::is_class< T > >::type * = 0, typename boost::disable_if< HasToString< T, std::string(T::*)() const > >::type * = 0) [private]

Forward all objects that do not implement a ToString() method.

21.123.3.4 template<typename T > void mlpack::util::PrefixedOutStream::CallBaseLogic (const T & *s*, typename boost::enable_if< boost::is_class< T > >::type * = 0, typename boost::enable_if< HasToString< T, std::string(T::*)() const > >::type * = 0) [private]

Call ToString() on all objects that implement ToString() before forwarding.

21.123.3.5 PrefixedOutStream& mlpack::util::PrefixedOutStream::operator<< (bool *val*)

Write a bool to the stream.

21.123.3.6 PrefixedOutStream& mlpack::util::PrefixedOutStream::operator<< (short *val*)

Write a short to the stream.

21.123.3.7 PrefixedOutStream& mlpack::util::PrefixedOutStream::operator<< (unsigned short *val*)

Write an unsigned short to the stream.

21.123.3.8 PrefixedOutStream& mlpack::util::PrefixedOutStream::operator<< (int *val*)

Write an int to the stream.

21.123.3.9 PrefixedOutStream& mlpack::util::PrefixedOutStream::operator<< (unsigned int *val*)

Write an unsigned int to the stream.

21.123.3.10 PrefixedOutStream& mlpack::util::PrefixedOutStream::operator<< (long *val*)

Write a long to the stream.

21.123.3.11 PrefixedOutStream& mlpack::util::PrefixedOutStream::operator<< (unsigned long *val*)

Write an unsigned long to the stream.

21.123.3.12 PrefixedOutStream& mlpack::util::PrefixedOutStream::operator<< (float *val*)

Write a float to the stream.

21.123.3.13 PrefixedOutStream& mlpack::util::PrefixedOutStream::operator<< (double *val*)

Write a double to the stream.

21.123.3.14 PrefixedOutStream& mlpack::util::PrefixedOutStream::operator<< (long double *val*)

Write a long double to the stream.

21.123.3.15 PrefixedOutStream& mlpack::util::PrefixedOutStream::operator<< (void * *val*)

Write a void pointer to the stream.

21.123.3.16 PrefixedOutStream& mlpack::util::PrefixedOutStream::operator<< (const char * *str*)

Write a character array to the stream.

21.123.3.17 PrefixedOutStream& mlpack::util::PrefixedOutStream::operator<< (std::string & *str*)

Write a string to the stream.

21.123.3.18 PrefixedOutStream& mlpack::util::PrefixedOutStream::operator<< (std::streambuf * *sb*)

Write a streambuf to the stream.

21.123.3.19 PrefixedOutStream& mlpack::util::PrefixedOutStream::operator<< (std::ostream &(*)(std::ostream &) *pf*)

Write an ostream manipulator function to the stream.

21.123.3.20 PrefixedOutStream& mlpack::util::PrefixedOutStream::operator<< (std::ios &(*)(std::ios &) *pf*)

Write an ios manipulator function to the stream.

21.123.3.21 PrefixedOutStream& mlpack::util::PrefixedOutStream::operator<< (std::ios_base &(*)(std::ios_base &) *pf*)

Write an ios_base manipulator function to the stream.

21.123.3.22 template<typename T > PrefixedOutStream& mlpack::util::PrefixedOutStream::operator<< (const T & *s*)

Write anything else to the stream.

21.123.3.23 void mlpack::util::PrefixedOutStream::PrefixIfNeeded ( ) [inline], [private]

Output the prefix, but only if we need to and if we are allowed to.

21.123.4 Member Data Documentation

21.123.4.1 bool mlpack::util::PrefixedOutStream::carriageReturned [private]

If true, the previous call to operator<< encountered a CR, and a prefix will be necessary.

Definition at line 183 of file prefixedoutstream.hpp.

21.123.4.2 std::ostream& mlpack::util::PrefixedOutStream::destination

The output stream that all data is to be sent too; example: std::cout.

Definition at line 128 of file prefixedoutstream.hpp.

21.123.4.3 bool mlpack::util::PrefixedOutStream::fatal [private]

If true, the application will terminate with an error code when a CR is encountered.

Definition at line 187 of file prefixedoutstream.hpp.

21.123.4.4 bool mlpack::util::PrefixedOutStream::ignoreInput

Discards input, prints nothing if true.

Definition at line 131 of file prefixedoutstream.hpp.

21.123.4.5 std::string mlpack::util::PrefixedOutStream::prefix [private]

Contains the prefix we must prepend to each line.

Definition at line 179 of file prefixedoutstream.hpp.

The documentation for this class was generated from the following file:

- src/mlpack/core/util/**prefixedoutstream.hpp**

21.124 mlpack::util::ProgramDoc Class Reference

A static object whose constructor registers program documentation with the **CLI** (p. 132) class.

Public Member Functions

- **ProgramDoc** (const std::string &**programName**, const std::string &**documentation**)

  *Construct a **ProgramDoc** (p. 538) object.*

Public Attributes

- std::string **documentation**

  Documentation for what the program does.
- std::string **programName**

  The name of the program.

21.124.1 Detailed Description

A static object whose constructor registers program documentation with the **CLI** (p. 132) class.

This should not be used outside of **CLI** (p. 132) itself, and you should use the **PROGRAM_INFO()** (p. 630) macro to declare these objects. Only one **ProgramDoc** (p. 538) object should ever exist.

See Also

core/io/cli.hpp, **mlpack::CLI** (p. 132)

Definition at line 90 of file option.hpp.

21.124.2 Constructor & Destructor Documentation

21.124.2.1 mlpack::util::ProgramDoc::ProgramDoc (const std::string & *programName,* const std::string & *documentation*)

Construct a **ProgramDoc** (p. 538) object.

When constructed, it will register itself with **CLI** (p. 132).

Parameters

programName	Short string representing the name of the program.
documentation	Long string containing documentation on how to use the program and what it is. No newline characters are necessary; this is taken care of by **CLI** (p. 132) later.

21.124.3 Member Data Documentation

21.124.3.1 std::string mlpack::util::ProgramDoc::documentation

Documentation for what the program does.

Definition at line 108 of file option.hpp.

21.124.3.2 std::string mlpack::util::ProgramDoc::programName

The name of the program.

Definition at line 106 of file option.hpp.

The documentation for this class was generated from the following file:

- src/mlpack/core/util/**option.hpp**

21.125 mlpack::util::SaveRestoreUtility Class Reference

Public Member Functions

- **SaveRestoreUtility** ()
- ∼**SaveRestoreUtility** ()
- template<typename T >
  T & **LoadParameter** (T &t, const std::string &name)

  LoadParameter loads a parameter from the parameters map.
- template<typename T >
  std::vector< T > & **LoadParameter** (std::vector< T > &v, const std::string &name)

  LoadParameter loads a parameter from the parameters map.
- char **LoadParameter** (char c, const std::string &name)

  LoadParameter loads a character from the parameters map.
- std::string **LoadParameter** (std::string &str, const std::string &name)

  LoadParameter loads a string from the parameters map.
- arma::mat & **LoadParameter** (arma::mat &matrix, const std::string &name)

  LoadParameter loads an arma::mat from the parameters map.
- template<>
  arma::vec & **LoadParameter** (arma::vec &t, const std::string &name)

  Specialization for arma::vec.
- bool **ReadFile** (const std::string &filename)

  ReadFile reads an XML tree from a file.
- template<typename T >
  void **SaveParameter** (const T &t, const std::string &name)

  SaveParameter saves a parameter to the parameters map.
- template<typename T >
  void **SaveParameter** (const std::vector< T > &v, const std::string &name)

  SaveParameter saves a parameter to the parameters map.
- void **SaveParameter** (const char c, const std::string &name)

  SaveParameter saves a character to the parameters map.
- void **SaveParameter** (const arma::mat &mat, const std::string &name)

  SaveParameter saves an arma::mat to the parameters map.
- template<>
  void **SaveParameter** (const arma::vec &t, const std::string &name)

  Specialization for arma::vec.
- bool **WriteFile** (const std::string &filename)

  WriteFile writes the XML tree to a file.

Private Member Functions

- void **RecurseOnNodes** (xmlNode ∗n)

  RecurseOnNodes performs a depth first search of the XML tree.

Private Attributes

- std::map< std::string, std::string > **parameters**

  parameters contains a list of names and parameters in string form.

21.125.1 Detailed Description

Definition at line 42 of file save_restore_utility.hpp.

21.125.2 Constructor & Destructor Documentation

21.125.2.1 mlpack::util::SaveRestoreUtility::SaveRestoreUtility ( ) [inline]

Definition at line 56 of file save_restore_utility.hpp.

21.125.2.2 mlpack::util::SaveRestoreUtility::~SaveRestoreUtility ( ) [inline]

Definition at line 57 of file save_restore_utility.hpp.

References parameters.

21.125.3 Member Function Documentation

21.125.3.1 template<typename T > T& mlpack::util::SaveRestoreUtility::LoadParameter (T & *t,* const std::string & *name*)

LoadParameter loads a parameter from the parameters map.

21.125.3.2 template<typename T > std::vector<T>& mlpack::util::SaveRestoreUtility::LoadParameter (std::vector< T > & *v,* const std::string & *name*)

LoadParameter loads a parameter from the parameters map.

21.125.3.3 char mlpack::util::SaveRestoreUtility::LoadParameter (char *c,* const std::string & *name*)

LoadParameter loads a character from the parameters map.

21.125.3.4 std::string mlpack::util::SaveRestoreUtility::LoadParameter (std::string & *str,* const std::string & *name*)

LoadParameter loads a string from the parameters map.

21.125.3.5 arma::mat& mlpack::util::SaveRestoreUtility::LoadParameter (arma::mat & *matrix,* const std::string & *name*)

LoadParameter loads an arma::mat from the parameters map.

21.125.3.6 template<> arma::vec& mlpack::util::SaveRestoreUtility::LoadParameter (arma::vec & *t,* const std::string & *name*)

Specialization for arma::vec.

21.125.3.7 bool mlpack::util::SaveRestoreUtility::ReadFile (const std::string & *filename*)

ReadFile reads an XML tree from a file.

21.125.3.8 void mlpack::util::SaveRestoreUtility::RecurseOnNodes (xmlNode * *n*) [private]

RecurseOnNodes performs a depth first search of the XML tree.

21.125.3.9 template<typename T > void mlpack::util::SaveRestoreUtility::SaveParameter (const T & *t,* const std::string & *name*)

SaveParameter saves a parameter to the parameters map.

21.125.3.10 template<typename T > void mlpack::util::SaveRestoreUtility::SaveParameter (const std::vector< T > & *v,* const std::string & *name*)

SaveParameter saves a parameter to the parameters map.

21.125.3.11 void mlpack::util::SaveRestoreUtility::SaveParameter (const char *c,* const std::string & *name*)

SaveParameter saves a character to the parameters map.

21.125.3.12 void mlpack::util::SaveRestoreUtility::SaveParameter (const arma::mat & *mat,* const std::string & *name*)

SaveParameter saves an arma::mat to the parameters map.

21.125.3.13 template<> void mlpack::util::SaveRestoreUtility::SaveParameter (const arma::vec & *t,* const std::string & *name*)

Specialization for arma::vec.

21.125.3.14 bool mlpack::util::SaveRestoreUtility::WriteFile (const std::string & *filename*)

WriteFile writes the XML tree to a file.

21.125.4 Member Data Documentation

21.125.4.1 std::map<std::string, std::string> mlpack::util::SaveRestoreUtility::parameters [private]

parameters contains a list of names and parameters in string form.

Definition at line 48 of file save_restore_utility.hpp.

Referenced by ~SaveRestoreUtility().

The documentation for this class was generated from the following file:

- src/mlpack/core/util/**save_restore_utility.hpp**

Chapter 22

File Documentation

22.1 doc/guide/build.hpp File Reference

22.2 doc/guide/iodoc.hpp File Reference

22.3 doc/guide/matrices.hpp File Reference

22.4 doc/guide/sample.hpp File Reference

22.5 doc/guide/timer.hpp File Reference

22.6 doc/tutorials/det/det.txt File Reference

Tutorial for how to perform density estimation with Density Estimation Trees (DET).

Functions

- f f **$V** (t)\f $is the volume of the node\f $t\f $and\f $\tilde
- see subsection cli_alt_reg_tut
  Alternate DET **regularization**
  The usual regularized error f
  $R_ **alpha** (t)\f $of a node\f $t\f $is given by

Variables

- f is the **set** of leaves in the
  subtree rooted at f $t f For
  the purposes of density **estimation**
- this option is not available
  in DET right **now**

- f is the **set** of leaves in the subtree rooted at f $t f For the purposes of density there is a different form of **regularization**
- f is the **set** of leaves in the subtree rooted at f $t f For the purposes of density there is a different form of we penalize the sum of the inverse of the volumes of the leaves With this very small volume nodes are discouraged unless the data actually warrants it **Thus**

22.6.1 Detailed Description

Tutorial for how to perform density estimation with Density Estimation Trees (DET).

Author

Parikshit Ram

Definition in file **det.txt**.

22.6.2 Function Documentation

22.6.2.1 f f $V (t)

Definition at line 374 of file det.txt.

22.6.2.2 see subsection cli_alt_reg_tut Alternate DET regularization The usual regularized error f $R_ alpha (t)

Definition at line 367 of file det.txt.

22.6.3 Variable Documentation

22.6.3.1 f is the set of leaves in the subtree rooted at f $t f For the purposes of density estimation

Definition at line 377 of file det.txt.

22.6.3.2 this option is not available in DET right now

Definition at line 364 of file det.txt.

22.6.3.3 f is the set of leaves in the subtree rooted at f $t f For the purposes of density there is a different form of we penalize the sum of the inverse of the volumes of the leaves With this regularization

Definition at line 377 of file det.txt.

22.6.3.4 f is the set of leaves in the subtree rooted at f $t f For the purposes of density there is a different form of we penalize the sum of the inverse of the volumes of the leaves With this very small volume nodes are discouraged unless the data actually warrants it Thus

Definition at line 377 of file det.txt.

22.7 doc/tutorials/emst/emst.txt File Reference

Tutorial for the Euclidean Minimum Spanning Tree algorithm.

22.7.1 Detailed Description

Tutorial for the Euclidean Minimum Spanning Tree algorithm.

Author

Bill March

Definition in file **emst.txt**.

22.8 doc/tutorials/fastmks/fastmks.txt File Reference

Tutorial for how to use FastMKS in mlpack.

22.8.1 Detailed Description

Tutorial for how to use FastMKS in mlpack.

Author

Ryan Curtin

Definition in file **fastmks.txt**.

22.9 doc/tutorials/kmeans/kmeans.txt File Reference

Tutorial for how to use k-means in mlpack.

22.9.1 Detailed Description

Tutorial for how to use k-means in mlpack.

Author

Ryan Curtin

Definition in file **kmeans.txt**.

22.10 doc/tutorials/linear_regression/linear_regression.txt File Reference

Tutorial for how to use the LinearRegression class.

22.10.1 Detailed Description

Tutorial for how to use the LinearRegression class.

Author

James Cline

Definition in file **linear_regression.txt**.

22.11 doc/tutorials/neighbor_search/neighbor_search.txt File Reference

Tutorial for how to use the NeighborSearch class.

22.11.1 Detailed Description

Tutorial for how to use the NeighborSearch class.

Author

Ryan Curtin

Definition in file **neighbor_search.txt**.

22.12 doc/tutorials/range_search/range_search.txt File Reference

Tutorial for how to use the RangeSearch class.

22.12.1 Detailed Description

Tutorial for how to use the RangeSearch class.

Author

Ryan Curtin

Definition in file **range_search.txt**.

22.13 doc/tutorials/tutorials.txt File Reference

List of MLPACK tutorials.

22.13.1 Detailed Description

List of MLPACK tutorials.

Author

Ryan Curtin

Definition in file **tutorials.txt**.

22.14 src/mlpack/CMakeLists.txt File Reference

Functions

- **include_directories** (..) **set**(MLPACK_SRCS $

22.14.1 Function Documentation

22.14.1.1 include_directories (..)

Definition at line 1 of file CMakeLists.txt.

22.15 src/mlpack/core/CMakeLists.txt File Reference

Functions

- **add_subdirectory** (${dir}) endforeach() **set**(MLPACK_SRCS $
- **set** (DIRS arma_extend data dists kernels math metrics optimizers tree util) foreach(dir $

22.15.1 Function Documentation

22.15.1.1 add_subdirectory (*${dir}*)

Definition at line 15 of file CMakeLists.txt.

22.15.1.2 set (DIRS arma_extend data dists kernels math metrics optimizers tree *util*)

Definition at line 2 of file CMakeLists.txt.

22.16 src/mlpack/core/data/CMakeLists.txt File Reference

Functions

- **set** (SOURCES load.hpp load_impl.hpp normalize_labels.hpp normalize_labels_impl.hpp save.hpp save_impl.-hpp) set(DIR_SRCS) foreach(file $
- **set** (DIR_SRCS ${DIR_SRCS}${CMAKE_CURRENT_SOURCE_DIR}/${file}) endforeach() set(MLPACK_SRCS $

22.16.1 Function Documentation

22.16.1.1 set (SOURCES load.hpp load_impl.hpp normalize_labels.hpp normalize_labels_impl.hpp save.hpp save_impl. *hpp*)

Definition at line 3 of file CMakeLists.txt.

22.16.1.2 set (DIR_SRCS ${DIR_SRCS}${CMAKE_CURRENT_SOURCE_DIR}/${file})

Definition at line 15 of file CMakeLists.txt.

22.17 src/mlpack/core/dists/CMakeLists.txt File Reference

Functions

- **set** (SOURCES discrete_distribution.hpp discrete_distribution.cpp gaussian_distribution.hpp gaussian_-distribution.cpp) set(DIR_SRCS) foreach(file $
- **set** (DIR_SRCS ${DIR_SRCS}${CMAKE_CURRENT_SOURCE_DIR}/${file}) endforeach() set(MLPACK_SRCS $

22.17.1 Function Documentation

22.17.1.1 set (SOURCES discrete_distribution.hpp discrete_distribution.cpp gaussian_distribution.hpp gaussian_distribution. *cpp*)

Definition at line 3 of file CMakeLists.txt.

22.17.1.2 set (DIR_SRCS ${DIR_SRCS}${CMAKE_CURRENT_SOURCE_DIR}/${file})

Definition at line 13 of file CMakeLists.txt.

22.18 src/mlpack/core/kernels/CMakeLists.txt File Reference

Functions

- **set** (SOURCES cosine_distance.hpp cosine_distance_impl.hpp epanechnikov_kernel.hpp epanechnikov_kernel-_impl.hpp epanechnikov_kernel.cpp example_kernel.hpp gaussian_kernel.hpp hyperbolic_tangent_kernel.hpp kernel_traits.hpp laplacian_kernel.hpp linear_kernel.hpp polynomial_kernel.hpp pspectrum_string_kernel.hpp pspectrum_string_kernel_impl.hpp pspectrum_string_kernel.cpp spherical_kernel.hpp triangular_kernel.hpp) set(DIR_SRCS) foreach(file $
- **set** (DIR_SRCS ${DIR_SRCS}${CMAKE_CURRENT_SOURCE_DIR}/${file}) endforeach() set(MLPACK_SRCS $

22.18.1 Function Documentation

22.18.1.1 set (SOURCES cosine_distance.hpp cosine_distance_impl.hpp epanechnikov_kernel.hpp epanechnikov_kernel_impl.hpp epanechnikov_kernel.cpp example_kernel.hpp gaussian_kernel.hpp hyperbolic_tangent_kernel.hpp kernel_traits.hpp laplacian_kernel.hpp linear_kernel.hpp polynomial_kernel.hpp pspectrum_string_kernel.hpp pspectrum_string_kernel_impl.hpp pspectrum_string_kernel.cpp spherical_kernel.hpp triangular_kernel. *hpp*)

Definition at line 3 of file CMakeLists.txt.

22.18.1.2 set (DIR_SRCS ${DIR_SRCS}${CMAKE_CURRENT_SOURCE_DIR}/${file})

Definition at line 26 of file CMakeLists.txt.

22.19 src/mlpack/core/math/CMakeLists.txt File Reference

Functions

- **set** (SOURCES clamp.hpp lin_alg.hpp lin_alg.cpp random.hpp random.cpp range.hpp range_impl.hpp round.hpp) set(DIR_SRCS) foreach(file $
- **set** (DIR_SRCS ${DIR_SRCS}${CMAKE_CURRENT_SOURCE_DIR}/${file}) endforeach() set(MLPACK_SRCS $

22.19.1 Function Documentation

22.19.1.1 set (SOURCES clamp.hpp lin_alg.hpp lin_alg.cpp random.hpp random.cpp range.hpp range_impl.hpp round. *hpp*)

Definition at line 3 of file CMakeLists.txt.

22.19.1.2 set (DIR_SRCS ${DIR_SRCS}${CMAKE_CURRENT_SOURCE_DIR}/${file})

Definition at line 17 of file CMakeLists.txt.

22.20 src/mlpack/core/metrics/CMakeLists.txt File Reference

Functions

- **set** (SOURCES ip_metric.hpp ip_metric_impl.hpp lmetric.hpp lmetric_impl.hpp mahalanobis_distance.hpp mahalanobis_distance_impl.hpp) set(DIR_SRCS) foreach(file $
- **set** (DIR_SRCS ${DIR_SRCS}${CMAKE_CURRENT_SOURCE_DIR}/${file}) endforeach() set(MLPACK_SRCS $

22.20.1 Function Documentation

22.20.1.1 set (SOURCES ip_metric.hpp ip_metric_impl.hpp lmetric.hpp lmetric_impl.hpp mahalanobis_distance.hpp mahalanobis_distance_impl. *hpp*)

Definition at line 3 of file CMakeLists.txt.

22.20.1.2 set (DIR_SRCS ${DIR_SRCS}${CMAKE_CURRENT_SOURCE_DIR}/${file})

Definition at line 15 of file CMakeLists.txt.

22.21 src/mlpack/core/optimizers/aug_lagrangian/CMakeLists.txt File Reference

Functions

- **set** (SOURCES aug_lagrangian.hpp aug_lagrangian_impl.hpp aug_lagrangian_function.hpp aug_lagrangian_-function_impl.hpp aug_lagrangian_test_functions.hpp aug_lagrangian_test_functions.cpp) set(DIR_SRCS) foreach(file $
- **set** (DIR_SRCS ${DIR_SRCS}${CMAKE_CURRENT_SOURCE_DIR}/${file}) endforeach() set(MLPACK_SRCS $

22.21.1 Function Documentation

22.21.1.1 set (SOURCES aug_lagrangian.hpp aug_lagrangian_impl.hpp aug_lagrangian_function.hpp aug_lagrangian_function_impl.hpp aug_lagrangian_test_functions.hpp aug_lagrangian_test_functions. *cpp*)

Definition at line 1 of file CMakeLists.txt.

22.21.1.2 set (DIR_SRCS ${DIR_SRCS}${CMAKE_CURRENT_SOURCE_DIR}/${file})

Definition at line 12 of file CMakeLists.txt.

22.22 src/mlpack/core/optimizers/CMakeLists.txt File Reference

Functions

- **add_subdirectory** (${dir}) endforeach() **set**(MLPACK_SRCS $
- **set** (DIRS aug_lagrangian lbfgs sgd) foreach(dir $

22.22.1 Function Documentation

22.22.1.1 add_subdirectory (*${dir}*)

Definition at line 8 of file CMakeLists.txt.

22.22.1.2 set (DIRS aug_lagrangian lbfgs *sgd*)

Definition at line 1 of file CMakeLists.txt.

22.23 src/mlpack/core/optimizers/lbfgs/CMakeLists.txt File Reference

Functions

- **set** (SOURCES lbfgs_impl.hpp lbfgs.hpp test_functions.hpp test_functions.cpp) set(DIR_SRCS) foreach(file $
- **set** (DIR_SRCS ${DIR_SRCS}${CMAKE_CURRENT_SOURCE_DIR}/${file}) endforeach() set(MLPACK_SRCS $

22.23.1 Function Documentation

22.23.1.1 set (SOURCES lbfgs_impl.hpp lbfgs.hpp test_functions.hpp test_functions. *cpp*)

Definition at line 1 of file CMakeLists.txt.

22.23.1.2 set (DIR_SRCS ${DIR_SRCS}${CMAKE_CURRENT_SOURCE_DIR}/${file})

Definition at line 10 of file CMakeLists.txt.

22.24 src/mlpack/core/optimizers/sgd/CMakeLists.txt File Reference

Functions

- **set** (SOURCES sgd.hpp sgd_impl.hpp test_function.hpp test_function.cpp) set(DIR_SRCS) foreach(file $
- **set** (DIR_SRCS ${DIR_SRCS}${CMAKE_CURRENT_SOURCE_DIR}/${file}) endforeach() set(MLPACK_SRCS $

22.24.1 Function Documentation

22.24.1.1 set (SOURCES sgd.hpp sgd_impl.hpp test_function.hpp test_function. *cpp*)

Definition at line 1 of file CMakeLists.txt.

22.24.1.2 set (DIR_SRCS ${DIR_SRCS}${CMAKE_CURRENT_SOURCE_DIR}/${file})

Definition at line 10 of file CMakeLists.txt.

22.25 src/mlpack/core/tree/CMakeLists.txt File Reference

Functions

- **set** (SOURCES ballbound.hpp ballbound_impl.hpp binary_space_tree/binary_space_tree.hpp binary_space_tree/binary_space_tree_impl.hpp binary_space_tree/dual_tree_traverser.hpp binary_space_tree/dual_tree_traverser_impl.hpp binary_space_tree/single_tree_traverser.hpp binary_space_tree/single_tree_traverser_impl.hpp binary_space_tree/traits.hpp bounds.hpp cosine_tree/cosine_tree_impl.hpp cosine_tree/cosine_tree.hpp cosine_tree/cosine_tree_builder.hpp cosine_tree/cosine_tree_builder_impl.hpp cover_tree/cover_tree.hpp cover_tree/cover_tree_impl.hpp cover_tree/first_point_is_root.hpp cover_tree/single_tree_traverser.hpp cover_tree/single_tree_traverser_impl.hpp cover_tree/dual_tree_traverser.hpp cover_tree/dual_tree_traverser_impl.hpp

cover_tree/traits.hpp hrectbound.hpp hrectbound_impl.hpp mrkd_statistic.hpp mrkd_statistic_impl.hpp mrkd_-statistic.cpp periodichrectbound.hpp periodichrectbound_impl.hpp statistic.hpp tree_traits.hpp) set(DIR_SRCS) foreach(file $

- **set** (DIR_SRCS ${DIR_SRCS}${CMAKE_CURRENT_SOURCE_DIR}/${file}) endforeach() set(MLPACK_SRCS $

22.25.1 Function Documentation

22.25.1.1 set (SOURCES ballbound.hpp ballbound_impl.hpp binary_space_tree/binary_space_tree.hpp binary_space_tree/binary_space_tree_impl.hpp binary_space_tree/dual_tree_traverser.hpp binary_space_tree/dual_-tree_traverser_impl.hpp binary_space_tree/single_tree_traverser.hpp binary_space_tree/single_tree_traverser_impl.hpp binary_space_tree/traits.hpp bounds.hpp cosine_tree/cosine_tree_impl.hpp cosine_tree/cosine_tree.hpp cosine_tree/cosine_tree_builder.hpp cosine_tree/cosine_tree_builder_impl.hpp cover_tree/cover_tree.hpp cover_tree/cover_tree_impl.hpp cover_tree/first_point_is_root.hpp cover_tree/single_tree_traverser.hpp cover_tree/single_tree_traverser_impl.hpp cover_tree/dual_tree_traverser.hpp cover_tree/dual_tree_traverser_impl.hpp cover_tree/traits.hpp hrectbound.hpp hrectbound_impl.hpp mrkd_statistic.hpp mrkd_statistic_impl.hpp mrkd_statistic.cpp periodichrectbound.hpp periodichrectbound_impl.hpp statistic.hpp tree_traits. *hpp*)

Definition at line 3 of file CMakeLists.txt.

22.25.1.2 set (DIR_SRCS ${DIR_SRCS}${CMAKE_CURRENT_SOURCE_DIR}/${file})

Definition at line 40 of file CMakeLists.txt.

22.26 src/mlpack/core/util/CMakeLists.txt File Reference

Functions

- **set** (SOURCES cli.hpp cli.cpp cli_deleter.hpp cli_deleter.cpp cli_impl.hpp log.hpp log.cpp nulloutstream.hpp option.hpp option.cpp option_impl.hpp prefixedoutstream.hpp prefixedoutstream.cpp prefixedoutstream_impl.-hpp save_restore_utility.hpp save_restore_utility.cpp save_restore_utility_impl.hpp sfinae_utility.hpp string_util.-hpp string_util.cpp timers.hpp timers.cpp version.hpp version.cpp) set(DIR_SRCS) foreach(file $
- **set** (DIR_SRCS ${DIR_SRCS}${CMAKE_CURRENT_SOURCE_DIR}/${file}) endforeach() set(MLPACK_SRCS $

22.26.1 Function Documentation

22.26.1.1 set (SOURCES cli.hpp cli.cpp cli_deleter.hpp cli_deleter.cpp cli_impl.hpp log.hpp log.cpp nulloutstream.hpp option.hpp option.cpp option_impl.hpp prefixedoutstream.hpp prefixedoutstream.cpp prefixedoutstream_impl.hpp save_restore_utility.hpp save_restore_utility.cpp save_restore_utility_impl.hpp sfinae_utility.hpp string_util.hpp string_util.cpp timers.hpp timers.cpp version.hpp version. *cpp*)

Definition at line 3 of file CMakeLists.txt.

22.26.1.2 set (DIR_SRCS ${DIR_SRCS}${CMAKE_CURRENT_SOURCE_DIR}/${file})

Definition at line 33 of file CMakeLists.txt.

22.27 src/mlpack/methods/cf/CMakeLists.txt File Reference

Functions

- **set** (SOURCES cf.hpp cf.cpp) set(DIR_SRCS) foreach(file $
- **set** (DIR_SRCS ${DIR_SRCS}${CMAKE_CURRENT_SOURCE_DIR}/${file}) endforeach() set(MLPACK_SRCS $

22.27.1 Function Documentation

22.27.1.1 set (SOURCES cf.hpp cf. *cpp*)

Definition at line 3 of file CMakeLists.txt.

22.27.1.2 set (DIR_SRCS ${DIR_SRCS}${CMAKE_CURRENT_SOURCE_DIR}/${file})

Definition at line 11 of file CMakeLists.txt.

22.28 src/mlpack/methods/CMakeLists.txt File Reference

Functions

- **add_subdirectory** (${dir}) endforeach() **set**(MLPACK_SRCS $
- **set** (DIRS cf det emst fastmks gmm hmm kernel_pca kmeans lars linear_regression local_coordinate_coding logistic_regression lsh naive_bayes nca neighbor_search nmf pca radical range_search rann sparse_coding) foreach(dir $

22.28.1 Function Documentation

22.28.1.1 add_subdirectory (*${dir}*)

Definition at line 29 of file CMakeLists.txt.

22.28.1.2 set (DIRS cf det emst fastmks gmm hmm kernel_pca kmeans lars linear_regression local_coordinate_coding logistic_regression lsh naive_bayes nca neighbor_search nmf pca radical range_search rann *sparse_coding*)

Definition at line 2 of file CMakeLists.txt.

22.29 src/mlpack/methods/det/CMakeLists.txt File Reference

Functions

- **set** (SOURCES dtree.hpp dtree.cpp dt_utils.hpp dt_utils.cpp) set(DIR_SRCS) foreach(file $
- **set** (DIR_SRCS ${DIR_SRCS}${CMAKE_CURRENT_SOURCE_DIR}/${file}) endforeach() set(MLPACK_SRCS $

22.29.1 Function Documentation

22.29.1.1 set (SOURCES dtree.hpp dtree.cpp dt_utils.hpp dt_utils. *cpp*)

Definition at line 4 of file CMakeLists.txt.

22.29.1.2 set (DIR_SRCS ${DIR_SRCS}${CMAKE_CURRENT_SOURCE_DIR}/${file})

Definition at line 17 of file CMakeLists.txt.

22.30 src/mlpack/methods/emst/CMakeLists.txt File Reference

Functions

- **set** (SOURCES union_find.hpp dtb.hpp dtb_impl.hpp dtb_rules.hpp dtb_rules_impl.hpp dtb_stat.hpp edge_pair.-hpp) set(DIR_SRCS) foreach(file $
- **set** (DIR_SRCS ${DIR_SRCS}${CMAKE_CURRENT_SOURCE_DIR}/${file}) endforeach() set(MLPACK_SRCS $

22.30.1 Function Documentation

22.30.1.1 set (SOURCES union_find.hpp dtb.hpp dtb_impl.hpp dtb_rules.hpp dtb_rules_impl.hpp dtb_stat.hpp edge_pair. *hpp*)

Definition at line 3 of file CMakeLists.txt.

22.30.1.2 set (DIR_SRCS ${DIR_SRCS}${CMAKE_CURRENT_SOURCE_DIR}/${file})

Definition at line 18 of file CMakeLists.txt.

22.31 src/mlpack/methods/fastmks/CMakeLists.txt File Reference

Functions

- **set** (SOURCES fastmks.hpp fastmks_impl.hpp fastmks_rules.hpp fastmks_rules_impl.hpp) set(DIR_SRCS) foreach(file $
- **set** (DIR_SRCS ${DIR_SRCS}${CMAKE_CURRENT_SOURCE_DIR}/${file}) endforeach() set(MLPACK_SRCS $

22.31.1 Function Documentation

22.31.1.1 set (SOURCES fastmks.hpp fastmks_impl.hpp fastmks_rules.hpp fastmks_rules_impl. *hpp*)

Definition at line 3 of file CMakeLists.txt.

22.31.1.2 set (DIR_SRCS ${DIR_SRCS}${CMAKE_CURRENT_SOURCE_DIR}/${file})

Definition at line 13 of file CMakeLists.txt.

22.32 src/mlpack/methods/gmm/CMakeLists.txt File Reference

Functions

- **set** (SOURCES gmm.hpp gmm_impl.hpp phi.hpp em_fit.hpp em_fit_impl.hpp no_constraint.hpp positive_definite-_constraint.hpp diagonal_constraint.hpp eigenvalue_ratio_constraint.hpp) set(DIR_SRCS) foreach(file $
- **set** (DIR_SRCS ${DIR_SRCS}${CMAKE_CURRENT_SOURCE_DIR}/${file}) endforeach() set(MLPACK_SRCS $

22.32.1 Function Documentation

22.32.1.1 set (SOURCES gmm.hpp gmm_impl.hpp phi.hpp em_fit.hpp em_fit_impl.hpp no_constraint.hpp positive_definite_constraint.hpp diagonal_constraint.hpp eigenvalue_ratio_constraint. *hpp*)

Definition at line 3 of file CMakeLists.txt.

22.32.1.2 set (DIR_SRCS ${DIR_SRCS}${CMAKE_CURRENT_SOURCE_DIR}/${file})

Definition at line 18 of file CMakeLists.txt.

22.33 src/mlpack/methods/hmm/CMakeLists.txt File Reference

Functions

- **set** (SOURCES hmm.hpp hmm_impl.hpp hmm_util.hpp hmm_util_impl.hpp) set(DIR_SRCS) foreach(file $
- **set** (DIR_SRCS ${DIR_SRCS}${CMAKE_CURRENT_SOURCE_DIR}/${file}) endforeach() set(MLPACK_SRCS $

22.33.1 Function Documentation

22.33.1.1 set (SOURCES hmm.hpp hmm_impl.hpp hmm_util.hpp hmm_util_impl. *hpp*)

Definition at line 3 of file CMakeLists.txt.

22.33.1.2 set (DIR_SRCS ${DIR_SRCS}${CMAKE_CURRENT_SOURCE_DIR}/${file})

Definition at line 13 of file CMakeLists.txt.

22.34 src/mlpack/methods/kernel_pca/CMakeLists.txt File Reference

Functions

- **set** (SOURCES kernel_pca.hpp kernel_pca_impl.hpp) set(DIR_SRCS) foreach(file $
- **set** (DIR_SRCS ${DIR_SRCS}${CMAKE_CURRENT_SOURCE_DIR}/${file}) endforeach() set(MLPACK_SRCS $

22.34.1 Function Documentation

22.34.1.1 set (SOURCES kernel_pca.hpp kernel_pca_impl. *hpp*)

Definition at line 3 of file CMakeLists.txt.

22.34.1.2 set (DIR_SRCS ${DIR_SRCS}${CMAKE_CURRENT_SOURCE_DIR}/${file})

Definition at line 11 of file CMakeLists.txt.

22.35 src/mlpack/methods/kmeans/CMakeLists.txt File Reference

Functions

- **set** (SOURCES allow_empty_clusters.hpp kmeans.hpp kmeans_impl.hpp max_variance_new_cluster.hpp max_-variance_new_cluster_impl.hpp random_partition.hpp refined_start.hpp refined_start_impl.hpp) set(DIR_SRCS) foreach(file $
- **set** (DIR_SRCS ${DIR_SRCS}${CMAKE_CURRENT_SOURCE_DIR}/${file}) endforeach() set(MLPACK_SRCS $

22.35.1 Function Documentation

22.35.1.1 set (SOURCES allow_empty_clusters.hpp kmeans.hpp kmeans_impl.hpp max_variance_new_cluster.hpp max_variance_new_cluster_impl.hpp random_partition.hpp refined_start.hpp refined_start_impl. *hpp*)

Definition at line 3 of file CMakeLists.txt.

22.35.1.2 set (DIR_SRCS ${DIR_SRCS}${CMAKE_CURRENT_SOURCE_DIR}/${file})

Definition at line 17 of file CMakeLists.txt.

22.36 src/mlpack/methods/lars/CMakeLists.txt File Reference

Functions

- **set** (SOURCES lars.hpp lars.cpp) set(DIR_SRCS) foreach(file $
- **set** (DIR_SRCS ${DIR_SRCS}${CMAKE_CURRENT_SOURCE_DIR}/${file}) endforeach() set(MLPACK_SRCS $

22.36.1 Function Documentation

22.36.1.1 set (SOURCES lars.hpp lars. *cpp*)

Definition at line 3 of file CMakeLists.txt.

22.36.1.2 set (DIR_SRCS ${DIR_SRCS}${CMAKE_CURRENT_SOURCE_DIR}/${file})

Definition at line 11 of file CMakeLists.txt.

22.37 src/mlpack/methods/linear_regression/CMakeLists.txt File Reference

Functions

- **set** (SOURCES linear_regression.hpp linear_regression.cpp) set(DIR_SRCS) foreach(file $
- **set** (DIR_SRCS ${DIR_SRCS}${CMAKE_CURRENT_SOURCE_DIR}/${file}) endforeach() set(MLPACK_SRCS $

22.37.1 Function Documentation

22.37.1.1 set (SOURCES linear_regression.hpp linear_regression. *cpp*)

Definition at line 4 of file CMakeLists.txt.

22.37.1.2 set (DIR_SRCS ${DIR_SRCS}${CMAKE_CURRENT_SOURCE_DIR}/${file})

Definition at line 12 of file CMakeLists.txt.

22.38 src/mlpack/methods/local_coordinate_coding/CMakeLists.txt File Reference

Functions

- **set** (SOURCES lcc.hpp lcc_impl.hpp) set(DIR_SRCS) foreach(file $
- **set** (DIR_SRCS ${DIR_SRCS}${CMAKE_CURRENT_SOURCE_DIR}/${file}) endforeach() set(MLPACK_SRCS $

22.38.1 Function Documentation

22.38.1.1 set (SOURCES lcc.hpp lcc_impl. *hpp*)

Definition at line 6 of file CMakeLists.txt.

22.38.1.2 set (DIR_SRCS ${DIR_SRCS}${CMAKE_CURRENT_SOURCE_DIR}/${file})

Definition at line 14 of file CMakeLists.txt.

22.39 src/mlpack/methods/logistic_regression/CMakeLists.txt File Reference

Functions

- **set** (SOURCES logistic_regression.hpp logistic_regression_impl.hpp logistic_regression_function.hpp logistic_-regression_function.cpp) set(DIR_SRCS) foreach(file $

- **set** (DIR_SRCS ${DIR_SRCS}${CMAKE_CURRENT_SOURCE_DIR}/${file}) endforeach() set(MLPACK_SRCS $

22.39.1 Function Documentation

22.39.1.1 set (SOURCES logistic_regression.hpp logistic_regression_impl.hpp logistic_regression_function.hpp logistic_regression_function. *cpp*)

Definition at line 4 of file CMakeLists.txt.

22.39.1.2 set (DIR_SRCS ${DIR_SRCS}${CMAKE_CURRENT_SOURCE_DIR}/${file})

Definition at line 14 of file CMakeLists.txt.

22.40 src/mlpack/methods/lsh/CMakeLists.txt File Reference

Functions

- **set** (SOURCES lsh_search.hpp lsh_search_impl.hpp) set(DIR_SRCS) foreach(file $
- **set** (DIR_SRCS ${DIR_SRCS}${CMAKE_CURRENT_SOURCE_DIR}/${file}) endforeach() set(MLPACK_CONTRIB_SRCS $

22.40.1 Function Documentation

22.40.1.1 set (SOURCES lsh_search.hpp lsh_search_impl. *hpp*)

Definition at line 3 of file CMakeLists.txt.

22.40.1.2 set (DIR_SRCS ${DIR_SRCS}${CMAKE_CURRENT_SOURCE_DIR}/${file})

Definition at line 12 of file CMakeLists.txt.

22.41 src/mlpack/methods/naive_bayes/CMakeLists.txt File Reference

Functions

- **set** (SOURCES naive_bayes_classifier.hpp naive_bayes_classifier_impl.hpp) set(DIR_SRCS) foreach(file $
- **set** (DIR_SRCS ${DIR_SRCS}${CMAKE_CURRENT_SOURCE_DIR}/${file}) endforeach() set(MLPACK_SRCS $

22.41.1 Function Documentation

22.41.1.1 set (SOURCES naive_bayes_classifier.hpp naive_bayes_classifier_impl. *hpp*)

Definition at line 3 of file CMakeLists.txt.

22.41.1.2 set (DIR_SRCS ${DIR_SRCS}${CMAKE_CURRENT_SOURCE_DIR}/${file})

Definition at line 11 of file CMakeLists.txt.

22.42 src/mlpack/methods/nca/CMakeLists.txt File Reference

Functions

- **set** (SOURCES nca.hpp nca_impl.hpp nca_softmax_error_function.hpp nca_softmax_error_function_impl.hpp) set(DIR_SRCS) foreach(file $
- **set** (DIR_SRCS ${DIR_SRCS}${CMAKE_CURRENT_SOURCE_DIR}/${file}) endforeach() set(MLPACK_SRCS $

22.42.1 Function Documentation

22.42.1.1 set (SOURCES nca.hpp nca_impl.hpp nca_softmax_error_function.hpp nca_softmax_error_function_impl. *hpp*)

Definition at line 3 of file CMakeLists.txt.

22.42.1.2 set (DIR_SRCS ${DIR_SRCS}${CMAKE_CURRENT_SOURCE_DIR}/${file})

Definition at line 13 of file CMakeLists.txt.

22.43 src/mlpack/methods/neighbor_search/CMakeLists.txt File Reference

Functions

- **set** (SOURCES neighbor_search.hpp neighbor_search_impl.hpp neighbor_search_rules.hpp neighbor_search_rules_impl.hpp neighbor_search_stat.hpp sort_policies/nearest_neighbor_sort.hpp sort_policies/nearest_neighbor_sort.cpp sort_policies/nearest_neighbor_sort_impl.hpp sort_policies/furthest_neighbor_sort.hpp sort_policies/furthest_neighbor_sort.cpp sort_policies/furthest_neighbor_sort_impl.hpp typedef.hpp unmap.hpp unmap.cpp) set(DIR_SRCS) foreach(file $
- **set** (DIR_SRCS ${DIR_SRCS}${CMAKE_CURRENT_SOURCE_DIR}/${file}) endforeach() set(MLPACK_SRCS $

22.43.1 Function Documentation

22.43.1.1 set (SOURCES neighbor_search.hpp neighbor_search_impl.hpp neighbor_search_rules.hpp neighbor_search_rules_impl.hpp neighbor_search_stat.hpp sort_policies/nearest_neighbor_sort.hpp sort_policies/nearest_neighbor_sort.cpp sort_policies/nearest_neighbor_sort_impl.hpp sort_policies/furthest_neighbor_sort.hpp sort_policies/furthest_neighbor_sort.cpp sort_policies/furthest_neighbor_sort_impl.hpp typedef.hpp unmap.hpp unmap. *cpp*)

Definition at line 3 of file CMakeLists.txt.

22.43.1.2 set (DIR_SRCS ${DIR_SRCS}${CMAKE_CURRENT_SOURCE_DIR}/${file})

Definition at line 23 of file CMakeLists.txt.

22.44 src/mlpack/methods/nmf/CMakeLists.txt File Reference

Functions

- **set** (SOURCES mult_dist_update_rules.hpp mult_div_update_rules.hpp als_update_rules.hpp random_init.hpp random_acol_init.hpp nmf.hpp nmf_impl.hpp) set(DIR_SRCS) foreach(file $
- **set** (DIR_SRCS ${DIR_SRCS}${CMAKE_CURRENT_SOURCE_DIR}/${file}) endforeach() set(MLPACK_SRCS $

22.44.1 Function Documentation

22.44.1.1 set (SOURCES mult_dist_update_rules.hpp mult_div_update_rules.hpp als_update_rules.hpp random_init.hpp random_acol_init.hpp nmf.hpp nmf_impl. *hpp*)

Definition at line 3 of file CMakeLists.txt.

22.44.1.2 set (DIR_SRCS ${DIR_SRCS}${CMAKE_CURRENT_SOURCE_DIR}/${file})

Definition at line 16 of file CMakeLists.txt.

22.45 src/mlpack/methods/pca/CMakeLists.txt File Reference

Functions

- **set** (SOURCES pca.hpp pca.cpp) set(DIR_SRCS) foreach(file $
- **set** (DIR_SRCS ${DIR_SRCS}${CMAKE_CURRENT_SOURCE_DIR}/${file}) endforeach() set(MLPACK_SRCS $

22.45.1 Function Documentation

22.45.1.1 set (SOURCES pca.hpp pca. *cpp*)

Definition at line 3 of file CMakeLists.txt.

22.45.1.2 set (DIR_SRCS ${DIR_SRCS}${CMAKE_CURRENT_SOURCE_DIR}/${file})

Definition at line 11 of file CMakeLists.txt.

22.46 src/mlpack/methods/radical/CMakeLists.txt File Reference

Functions

- **set** (SOURCES radical.hpp radical.cpp) set(DIR_SRCS) foreach(file $
- **set** (DIR_SRCS ${DIR_SRCS}${CMAKE_CURRENT_SOURCE_DIR}/${file}) endforeach() set(MLPACK_SRCS $

22.46.1 Function Documentation

22.46.1.1 set (SOURCES radical.hpp radical. *cpp*)

Definition at line 3 of file CMakeLists.txt.

22.46.1.2 set (DIR_SRCS ${DIR_SRCS}${CMAKE_CURRENT_SOURCE_DIR}/${file})

Definition at line 11 of file CMakeLists.txt.

22.47 src/mlpack/methods/range_search/CMakeLists.txt File Reference

Functions

- **set** (SOURCES range_search.hpp range_search_impl.hpp range_search_rules.hpp range_search_rules_impl.hpp range_search_stat.hpp) set(DIR_SRCS) foreach(file $
- **set** (DIR_SRCS ${DIR_SRCS}${CMAKE_CURRENT_SOURCE_DIR}/${file}) endforeach() set(MLPACK_SRCS $

22.47.1 Function Documentation

22.47.1.1 set (SOURCES range_search.hpp range_search_impl.hpp range_search_rules.hpp range_search_rules_impl.hpp range_search_stat. *hpp*)

Definition at line 3 of file CMakeLists.txt.

22.47.1.2 set (DIR_SRCS ${DIR_SRCS}${CMAKE_CURRENT_SOURCE_DIR}/${file})

Definition at line 14 of file CMakeLists.txt.

22.48 src/mlpack/methods/rann/CMakeLists.txt File Reference

Functions

- **set** (SOURCES ra_search.hpp ra_search_impl.hpp ra_search_rules.hpp ra_search_rules_impl.hpp ra_typedef.hpp) set(DIR_SRCS) foreach(file $
- **set** (DIR_SRCS ${DIR_SRCS}${CMAKE_CURRENT_SOURCE_DIR}/${file}) endforeach() set(MLPACK_CONTRIB_SRCS $

22.48.1 Function Documentation

22.48.1.1 set (SOURCES ra_search.hpp ra_search_impl.hpp ra_search_rules.hpp ra_search_rules_impl.hpp ra_typedef. *hpp*)

Definition at line 4 of file CMakeLists.txt.

22.48.1.2 set (DIR_SRCS ${DIR_SRCS}${CMAKE_CURRENT_SOURCE_DIR}/${file})

Definition at line 21 of file CMakeLists.txt.

22.49 src/mlpack/methods/sparse_coding/CMakeLists.txt File Reference

Functions

- **set** (SOURCES data_dependent_random_initializer.hpp nothing_initializer.hpp random_initializer.hpp sparse_coding.hpp sparse_coding_impl.hpp) set(DIR_SRCS) foreach(file $
- **set** (DIR_SRCS ${DIR_SRCS}${CMAKE_CURRENT_SOURCE_DIR}/${file}) endforeach() set(MLPACK_SRCS $

22.49.1 Function Documentation

22.49.1.1 set (SOURCES data_dependent_random_initializer.hpp nothing_initializer.hpp random_initializer.hpp sparse_coding.hpp sparse_coding_impl. *hpp*)

Definition at line 3 of file CMakeLists.txt.

22.49.1.2 set (DIR_SRCS ${DIR_SRCS}${CMAKE_CURRENT_SOURCE_DIR}/${file})

Definition at line 14 of file CMakeLists.txt.

22.50 src/mlpack/tests/CMakeLists.txt File Reference

Functions

- **add_custom_command** (TARGET mlpack_test POST_BUILD COMMAND ${CMAKE_COMMAND}-E copy_directory ${CMAKE_CURRENT_SOURCE_DIR}/data/${PROJECT_BINARY_DIR}) add_custom_command(TARGET mlpack_test POST_BUILD COMMAND $
- **add_executable** (mlpack_test mlpack_test.cpp allkfn_test.cpp allknn_test.cpp allkrann_search_test.cpp arma_extend_test.cpp aug_lagrangian_test.cpp cf_test.cpp cli_test.cpp det_test.cpp distribution_test.cpp emst_test.cpp fastmks_test.cpp gmm_test.cpp hmm_test.cpp kernel_test.cpp kernel_pca_test.cpp kernel_traits_test.cpp kmeans_test.cpp lars_test.cpp lbfgs_test.cpp lin_alg_test.cpp linear_regression_test.cpp load_save_test.cpp local_coordinate_coding_test.cpp logistic_regression_test.cpp lrsdp_test.cpp lsh_test.cpp math_test.cpp metric_test.cpp nbc_test.cpp nca_test.cpp nmf_test.cpp pca_test.cpp radical_test.cpp range_search_test.cpp save_restore_utility_test.cpp sgd_test.cpp sort_policy_test.cpp sparse_coding_test.cpp tree_test.cpp tree_traits_test.cpp union_find_test.cpp) target_link_libraries(mlpack_test mlpack $

22.50.1 Function Documentation

22.50.1.1 add_custom_command (TARGET mlpack_test POST_BUILD COMMAND ${CMAKE_COMMAND}-E copy_directory ${CMAKE_CURRENT_SOURCE_DIR}/data/${PROJECT_BINARY_DIR})

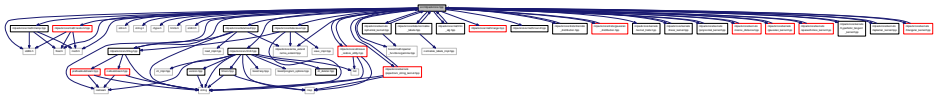
Definition at line 53 of file CMakeLists.txt.

22.50.1.2 add_executable (mlpack_test mlpack_test.cpp allkfn_test.cpp allknn_test.cpp allkrann_search_test.cpp arma_extend_test.cpp aug_lagrangian_test.cpp cf_test.cpp cli_test.cpp det_test.cpp distribution_test.cpp emst_test.cpp fastmks_test.cpp gmm_test.cpp hmm_test.cpp kernel_test.cpp kernel_pca_test.cpp kernel_traits_test.cpp kmeans_test.cpp lars_test.cpp lbfgs_test.cpp lin_alg_test.cpp linear_regression_test.cpp load_save_test.cpp local_coordinate_coding_test.cpp logistic_regression_test.cpp lrsdp_test.cpp lsh_test.cpp math_test.cpp metric_test.cpp nbc_test.cpp nca_test.cpp nmf_test.cpp pca_test.cpp radical_test.cpp range_search_test.cpp save_restore_utility_test.cpp sgd_test.cpp sort_policy_test.cpp sparse_coding_test.cpp tree_test.cpp tree_traits_test.cpp union_find_test. *cpp*)

Definition at line 2 of file CMakeLists.txt.

22.51 src/mlpack/core.hpp File Reference

Include dependency graph for core.hpp:



This graph shows which files directly or indirectly include this file:

Macros

- #define **_USE_MATH_DEFINES**
- #define **force_inline**
- #define **M_PI** 3.14159265358979323846264338327 9

22.51.1 Macro Definition Documentation

22.51.1.1 #define _USE_MATH_DEFINES

Definition at line 163 of file core.hpp.

22.51.1.2 #define force_inline

Definition at line 175 of file core.hpp.

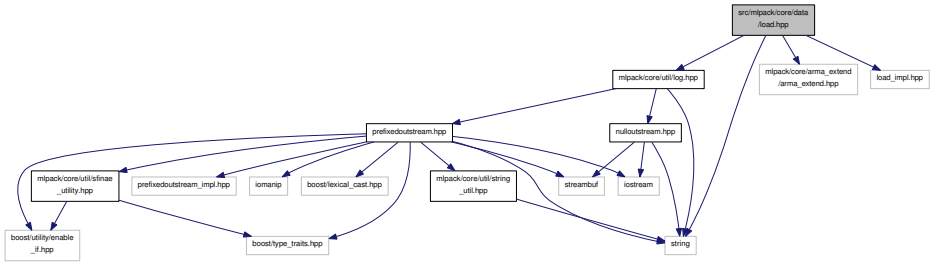
22.51.1.3 #define M_PI 3.14159265358979323846264338327 9

Definition at line 171 of file core.hpp.

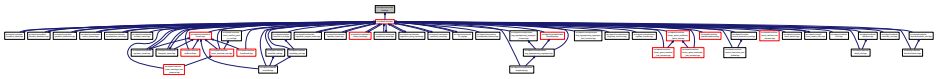
Referenced by mlpack::kernel::SphericalKernel::Normalizer(), mlpack::kernel::GaussianKernel::Normalizer(), and mlpack::gmm::phi().

22.52 src/mlpack/core/data/load.hpp File Reference

Include dependency graph for load.hpp:



This graph shows which files directly or indirectly include this file:



Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::data**

  Functions to load and save matrices.

Functions

- template<typename eT >
  bool **mlpack::data::Load** (const std::string &filename, arma::Mat< eT > &matrix, bool fatal=false, bool transpose=true)

  Loads a matrix from file, guessing the filetype from the extension.

22.52.1 Detailed Description

Author

Ryan Curtin

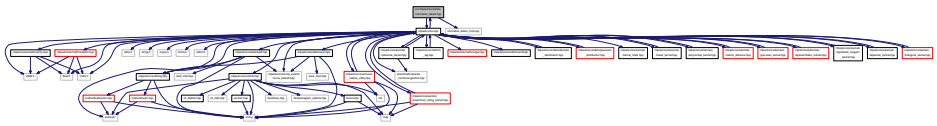
Load an Armadillo matrix from file. This is necessary because Armadillo does not transpose matrices on input, and it allows us to give better error output.

This file is part of MLPACK 1.0.8.

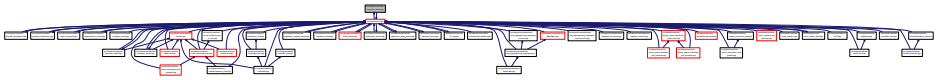
Definition in file **load.hpp**.

22.53 src/mlpack/core/data/normalize_labels.hpp File Reference

Include dependency graph for normalize_labels.hpp:



This graph shows which files directly or indirectly include this file:



Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::data**

  Functions to load and save matrices.

Functions

- template<typename eT >
  void **mlpack::data::NormalizeLabels** (const arma::Col< eT > &labelsIn, arma::Col< size_t > &labels, arma::Col< eT > &mapping)

  Given a set of labels of a particular datatype, convert them to unsigned labels in the range [0, n) where n is the number of different labels.

- template<typename eT >
  void **mlpack::data::RevertLabels** (const arma::Col< size_t > &labels, const arma::Col< eT > &mapping, arma::Col< eT > &labelsOut)

  Given a set of labels that have been mapped to the range [0, n), map them back to the original labels given by the 'mapping' vector.

22.53.1 Detailed Description

Author

Ryan Curtin

Often labels are not given as {0, 1, 2, ...} but instead {1, 2, ...} or even {-1, 1} or otherwise. The purpose of this function is to normalize labels to {0, 1, 2, ...} and provide a mapping back to those labels.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

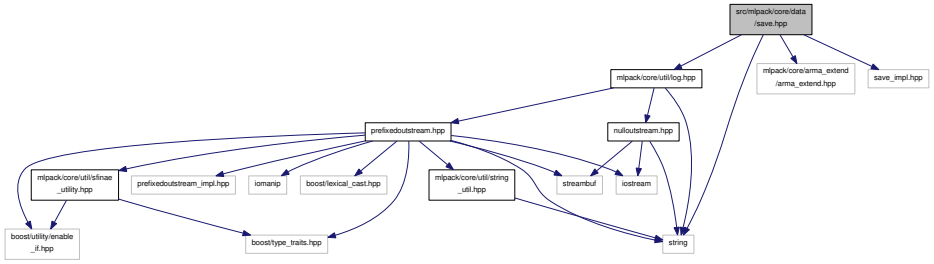
MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

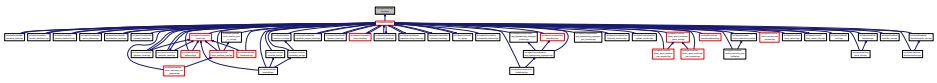
Definition in file **normalize_labels.hpp**.

22.54 src/mlpack/core/data/save.hpp File Reference

Include dependency graph for save.hpp:



This graph shows which files directly or indirectly include this file:



Namespaces

- **mlpack**

Linear algebra utility functions, generally performed on matrices or vectors.

- **mlpack::data**

  Functions to load and save matrices.

Functions

- template<typename eT >
  bool **mlpack::data::Save** (const std::string &filename, const arma::Mat< eT > &matrix, bool fatal=false, bool transpose=true)

  Saves a matrix to file, guessing the filetype from the extension.

22.54.1 Detailed Description

Author

Ryan Curtin

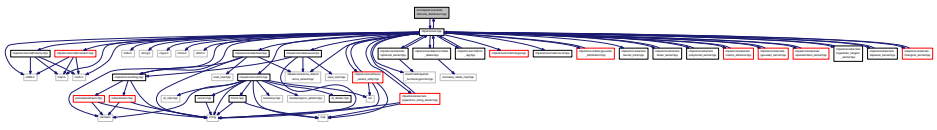
Save an Armadillo matrix to file. This is necessary because Armadillo does not transpose matrices upon saving, and it allows us to give better error output.

This file is part of MLPACK 1.0.8.

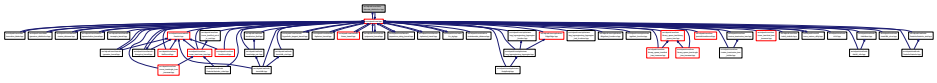
Definition in file **save.hpp**.

22.55 src/mlpack/core/dists/discrete_distribution.hpp File Reference

Include dependency graph for discrete_distribution.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class **mlpack::distribution::DiscreteDistribution**

  A discrete distribution where the only observations are discrete observations.

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::distribution**

  Probability distributions.

22.55.1 Detailed Description

Author

Ryan Curtin

Implementation of the discrete distribution, where each discrete observation has a given probability.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

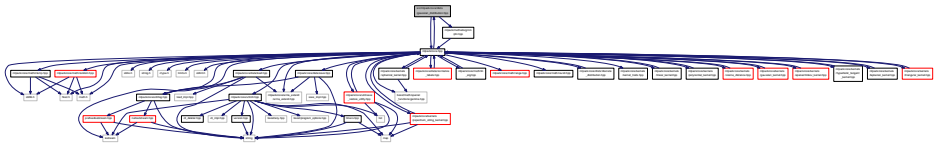
MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.-gnu.org/licenses/`.

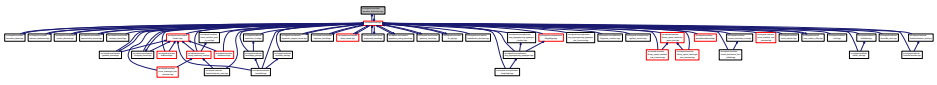
Definition in file **discrete_distribution.hpp**.

22.56 src/mlpack/core/dists/gaussian_distribution.hpp File Reference

Include dependency graph for gaussian_distribution.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class **mlpack::distribution::GaussianDistribution**

  A single multivariate Gaussian distribution.

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::distribution**

  Probability distributions.

22.56.1 Detailed Description

Author

Ryan Curtin

Implementation of the Gaussian distribution.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

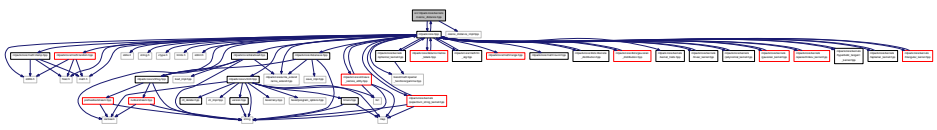
MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

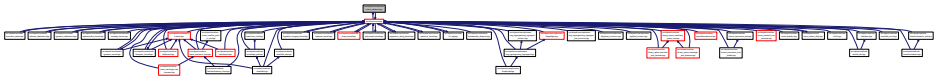
Definition in file **gaussian_distribution.hpp**.

22.57 src/mlpack/core/kernels/cosine_distance.hpp File Reference

Include dependency graph for cosine_distance.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class **mlpack::kernel::CosineDistance**

  The cosine distance (or cosine similarity).
- class **mlpack::kernel::KernelTraits< CosineDistance >**

  Kernel traits for the cosine distance.

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::kernel**

  Kernel functions.

22.57.1 Detailed Description

Author

Ryan Curtin

This implements the cosine distance (or cosine similarity) between two vectors, which is a measure of the angle between the two vectors.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

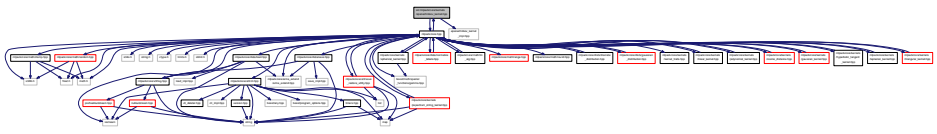
MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

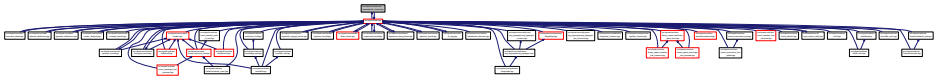
Definition in file **cosine_distance.hpp**.

22.58 src/mlpack/core/kernels/epanechnikov_kernel.hpp File Reference

Include dependency graph for epanechnikov_kernel.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class **mlpack::kernel::EpanechnikovKernel**

  The Epanechnikov kernel, defined as.
- class **mlpack::kernel::KernelTraits< EpanechnikovKernel >**

  Kernel traits for the Epanechnikov kernel.

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::kernel**

  Kernel functions.

22.58.1 Detailed Description

Author

Neil Slagle

Definition of the Epanechnikov kernel.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

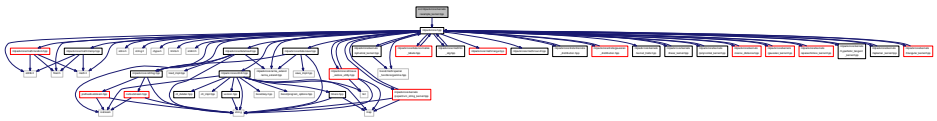
MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

Definition in file **epanechnikov_kernel.hpp**.

22.59 src/mlpack/core/kernels/example_kernel.hpp File Reference

Include dependency graph for example_kernel.hpp:



Classes

- class **mlpack::kernel::ExampleKernel**

  An example kernel function.

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::kernel**

  Kernel functions.

22.59.1 Detailed Description

Author

Ryan Curtin

This is an example kernel. If you are making your own kernel, follow the outline specified in this file.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

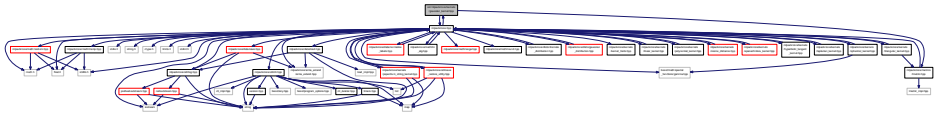
MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

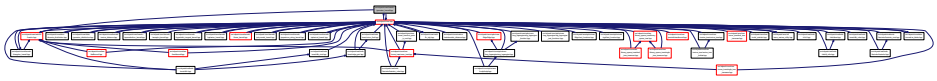
Definition in file **example_kernel.hpp**.

22.60 src/mlpack/core/kernels/gaussian_kernel.hpp File Reference

Include dependency graph for gaussian_kernel.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class **mlpack::kernel::GaussianKernel**

  The standard Gaussian kernel.
- class **mlpack::kernel::KernelTraits**< **GaussianKernel** >

  Kernel traits for the Gaussian kernel.

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::kernel**

  Kernel functions.

22.60.1 Detailed Description

Author

Wei Guan
James Cline
Ryan Curtin

Implementation of the Gaussian kernel (GaussianKernel).

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

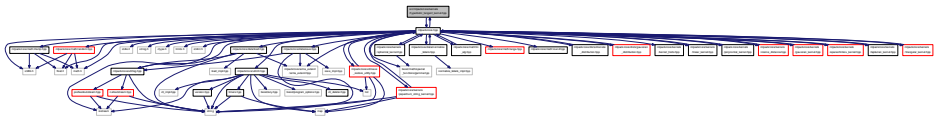
MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

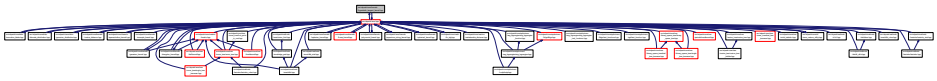
Definition in file **gaussian_kernel.hpp**.

22.61 src/mlpack/core/kernels/hyperbolic_tangent_kernel.hpp File Reference

Include dependency graph for hyperbolic_tangent_kernel.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class **mlpack::kernel::HyperbolicTangentKernel**

  Hyperbolic tangent kernel.

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::kernel**

  Kernel functions.

22.61.1 Detailed Description

Author

Ajinkya Kale `kaleajinkya@gmail.com`

Implementation of the hyperbolic tangent kernel.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

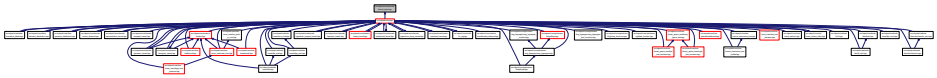
MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

Definition in file **hyperbolic_tangent_kernel.hpp**.

22.62 src/mlpack/core/kernels/kernel_traits.hpp File Reference

This graph shows which files directly or indirectly include this file:



Classes

- class **mlpack::kernel::KernelTraits**< **KernelType** >

  This is a template class that can provide information about various kernels.

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::kernel**

  Kernel functions.

22.62.1 Detailed Description

Author

Ryan Curtin

This provides the KernelTraits class, a template class to get information about various kernels.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

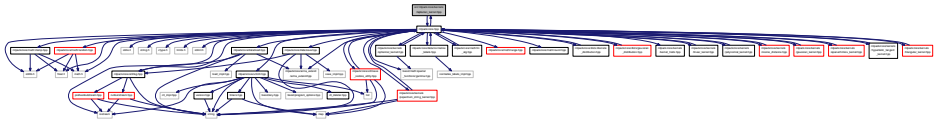
MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

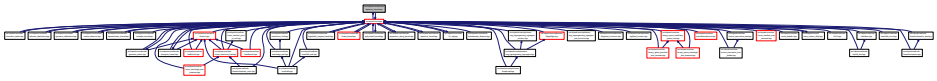
Definition in file **kernel_traits.hpp**.

22.63 src/mlpack/core/kernels/laplacian_kernel.hpp File Reference

Include dependency graph for laplacian_kernel.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class **mlpack::kernel::KernelTraits**< **LaplacianKernel** >

  Kernel traits of the Laplacian kernel.
- class **mlpack::kernel::LaplacianKernel**

  The standard Laplacian kernel.

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::kernel**

  Kernel functions.

22.63.1 Detailed Description

Author

Ajinkya Kale `kaleajinkya@gmail.com`

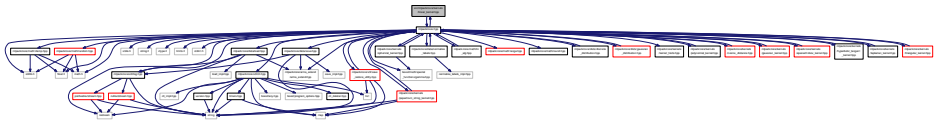
Implementation of the Laplacian kernel (LaplacianKernel).

This file is part of MLPACK 1.0.8.

Definition in file **laplacian_kernel.hpp**.

22.64 src/mlpack/core/kernels/linear_kernel.hpp File Reference

Include dependency graph for linear_kernel.hpp:



This graph shows which files directly or indirectly include this file:

Classes

- class **mlpack::kernel::LinearKernel**

  The simple linear kernel (dot product).

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::kernel**

  Kernel functions.

22.64.1 Detailed Description

Author

Wei Guan
James Cline
Ryan Curtin

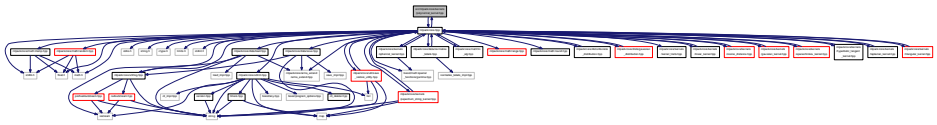
Implementation of the linear kernel (just the standard dot product).

This file is part of MLPACK 1.0.8.

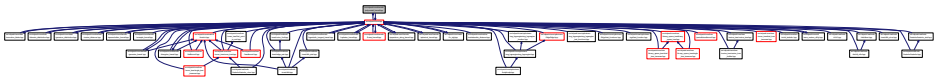
Definition in file **linear_kernel.hpp**.

22.65 src/mlpack/core/kernels/polynomial_kernel.hpp File Reference

Include dependency graph for polynomial_kernel.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class **mlpack::kernel::PolynomialKernel**

  The simple polynomial kernel.

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::kernel**

  Kernel functions.

22.65.1 Detailed Description

Author

Ajinkya Kale kaleajinkya@gmail.com

Implementation of the polynomial kernel (just the standard dot product).

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

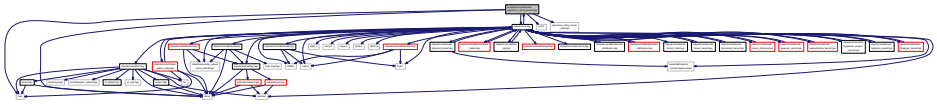
MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see http://www.-gnu.org/licenses/.

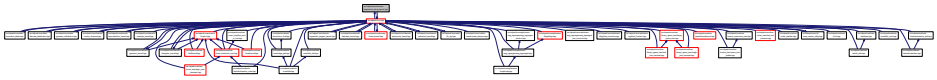
Definition in file **polynomial_kernel.hpp**.

22.66 src/mlpack/core/kernels/pspectrum_string_kernel.hpp File Reference

Include dependency graph for pspectrum_string_kernel.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class **mlpack::kernel::PSpectrumStringKernel**

  The p-spectrum string kernel.

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::kernel**

  Kernel functions.

22.66.1 Detailed Description

Author

Ryan Curtin

Implementation of the p-spectrum string kernel, created for use with FastMKS. Instead of passing a data matrix to FastMKS which stores the kernels, pass a one-dimensional data matrix (data vector) to FastMKS which stores indices of strings; then, the actual strings are given to the PSpectrumStringKernel at construction time, and the kernel knows to map the indices to actual strings.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

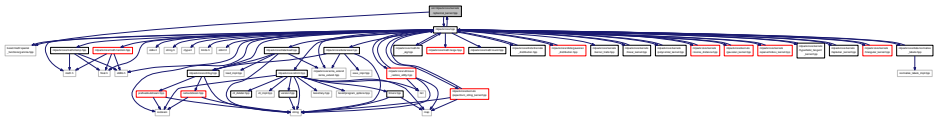
MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

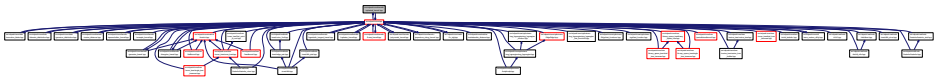
Definition in file **pspectrum_string_kernel.hpp**.

22.67 src/mlpack/core/kernels/spherical_kernel.hpp File Reference

Include dependency graph for spherical_kernel.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class **mlpack::kernel::KernelTraits< SphericalKernel >**
  Kernel traits for the spherical kernel.
- class **mlpack::kernel::SphericalKernel**

Namespaces

- **mlpack**
  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::kernel**
  Kernel functions.

22.67.1 Detailed Description

Author

Neil Slagle

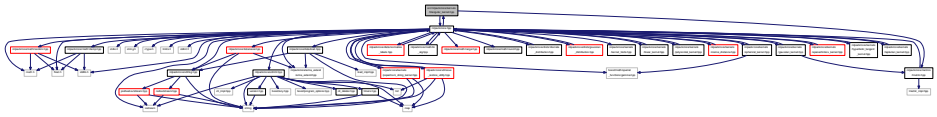
This is an example kernel. If you are making your own kernel, follow the outline specified in this file.

This file is part of MLPACK 1.0.8.

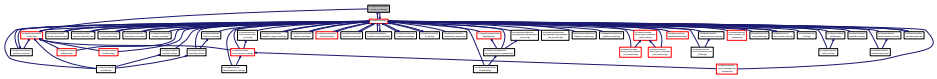
Definition in file **spherical_kernel.hpp**.

22.68 src/mlpack/core/kernels/triangular_kernel.hpp File Reference

Include dependency graph for triangular_kernel.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class **mlpack::kernel::KernelTraits**< **TriangularKernel** >

  Kernel traits for the triangular kernel.
- class **mlpack::kernel::TriangularKernel**

  The trivially simple triangular kernel, defined by.

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::kernel**

  Kernel functions.

22.68.1 Detailed Description

Author

Ryan Curtin

Definition and implementation of the trivially simple triangular kernel.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

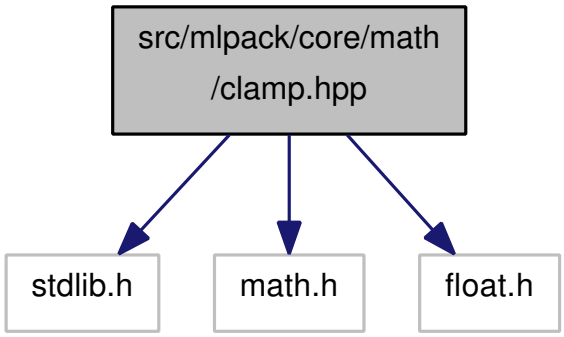
You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

Definition in file **triangular_kernel.hpp**.

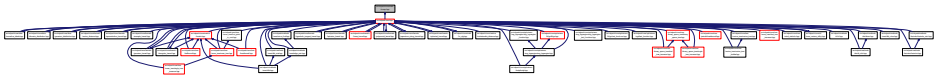
22.69 src/mlpack/core/math/clamp.hpp File Reference

Miscellaneous math clamping routines.

Include dependency graph for clamp.hpp:



This graph shows which files directly or indirectly include this file:



Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::math**

  Miscellaneous math routines.

Functions

- double **mlpack::math::ClampNonNegative** (const double d)

  Forces a number to be non-negative, turning negative numbers into zero.
- double **mlpack::math::ClampNonPositive** (const double d)

  Forces a number to be non-positive, turning positive numbers into zero.
- double **mlpack::math::ClampRange** (double value, const double rangeMin, const double rangeMax)

  Clamp a number between a particular range.

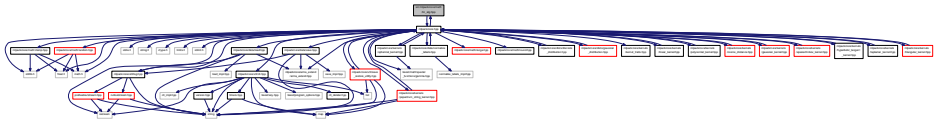
22.69.1 Detailed Description

Miscellaneous math clamping routines. This file is part of MLPACK 1.0.8.

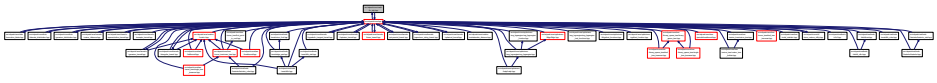
Definition in file **clamp.hpp**.

22.70 src/mlpack/core/math/lin_alg.hpp File Reference

Include dependency graph for lin_alg.hpp:



This graph shows which files directly or indirectly include this file:



Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::math**

  Miscellaneous math routines.

Functions

- void **mlpack::math::Center** (const arma::mat &x, arma::mat &xCentered)

  Creates a centered matrix, where centering is done by subtracting the sum over the columns (a column vector) from each column of the matrix.
- void **mlpack::math::Orthogonalize** (const arma::mat &x, arma::mat &W)

  Orthogonalize x and return the result in W, using eigendecomposition.
- void **mlpack::math::Orthogonalize** (arma::mat &x)

  Orthogonalize x in-place.
- void **mlpack::math::RandVector** (arma::vec &v)

  Overwrites a dimension-N vector to a random vector on the unit sphere in R^N.
- void **mlpack::math::RemoveRows** (const arma::mat &input, const std::vector< size_t > &rowsToRemove, arma::mat &output)

  Remove a certain set of rows in a matrix while copying to a second matrix.
- void **mlpack::math::VectorPower** (arma::vec &vec, double power)

  Auxiliary function to raise vector elements to a specific power.
- void **mlpack::math::WhitenUsingEig** (const arma::mat &x, arma::mat &xWhitened, arma::mat &whiteningMatrix)

  Whitens a matrix using the eigendecomposition of the covariance matrix.
- void **mlpack::math::WhitenUsingSVD** (const arma::mat &x, arma::mat &xWhitened, arma::mat &whiteningMatrix)

  Whitens a matrix using the singular value decomposition of the covariance matrix.

22.70.1 Detailed Description

Author

Nishant Mehta

Linear algebra utilities.

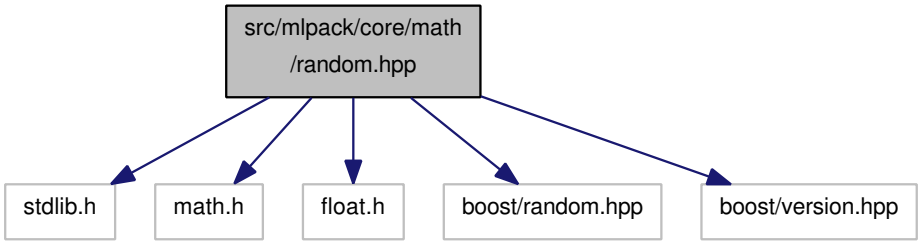
This file is part of MLPACK 1.0.8.

Definition in file **lin_alg.hpp**.

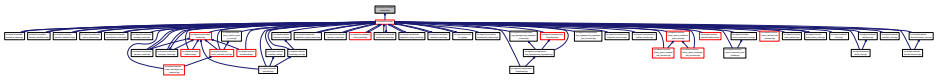
22.71 src/mlpack/core/math/random.hpp File Reference

Miscellaneous math random-related routines.

Include dependency graph for random.hpp:



This graph shows which files directly or indirectly include this file:



Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::math**

  Miscellaneous math routines.

Functions

- int **mlpack::math::RandInt** (const int hiExclusive)

  Generates a uniform random integer.
- int **mlpack::math::RandInt** (const int lo, const int hiExclusive)

  Generates a uniform random integer.
- double **mlpack::math::RandNormal** ()

  Generates a normally distributed random number with mean 0 and variance 1.
- double **mlpack::math::RandNormal** (const double mean, const double variance)

  Generates a normally distributed random number with specified mean and variance.
- double **mlpack::math::Random** ()

  Generates a uniform random number between 0 and 1.
- double **mlpack::math::Random** (const double lo, const double hi)

  Generates a uniform random number in the specified range.
- void **mlpack::math::RandomSeed** (const size_t seed)

  *Set the random seed used by the random functions (**Random()** (p. 99) and **RandInt()** (p. 98)).*

Variables

- boost::mt19937 **mlpack::math::randGen**
- boost::normal_distribution **mlpack::math::randNormalDist**
- boost::uniform_01
  < boost::mt19937, double > **mlpack::math::randUniformDist**

22.71.1 Detailed Description

Miscellaneous math random-related routines. This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

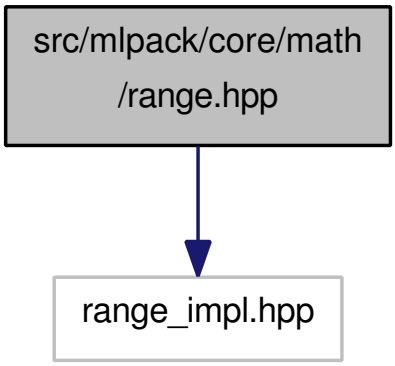
You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

Definition in file **random.hpp**.

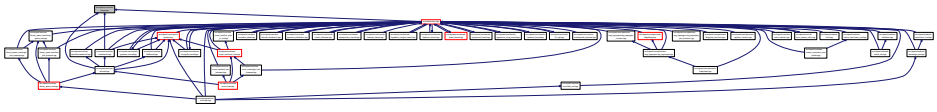
22.72 src/mlpack/core/math/range.hpp File Reference

Definition of the Range class, which represents a simple range with a lower and upper bound.

Include dependency graph for range.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class **mlpack::math::Range**

  Simple real-valued range.

Namespaces

- **mlpack**

Linear algebra utility functions, generally performed on matrices or vectors.

- **mlpack::math**

  Miscellaneous math routines.

22.72.1 Detailed Description

Definition of the Range class, which represents a simple range with a lower and upper bound. This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

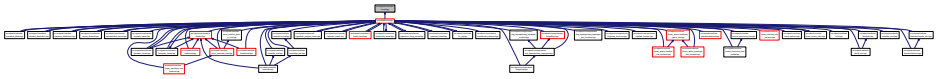
MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

Definition in file **range.hpp**.

22.73 src/mlpack/core/math/round.hpp File Reference

This graph shows which files directly or indirectly include this file:



22.73.1 Detailed Description

Author

Ryan Curtin

Implementation of round() for use on Visual Studio, where C99 isn't implemented.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

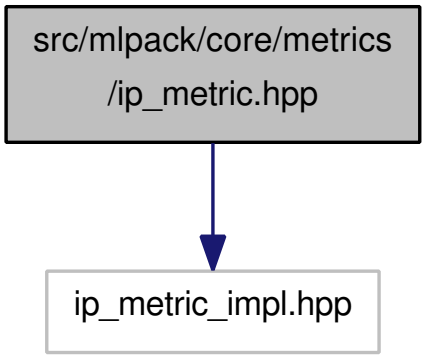
MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

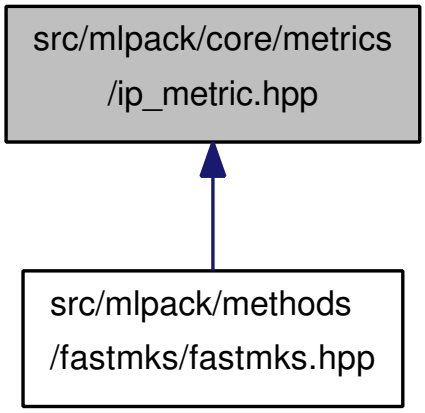
Definition in file **round.hpp**.

22.74 src/mlpack/core/metrics/ip_metric.hpp File Reference

Include dependency graph for ip_metric.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class **mlpack::metric::IPMetric**< **KernelType** >

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::metric**

22.74.1 Detailed Description

Author

Ryan Curtin

Inner product induced metric. If given a kernel function, this gives the complementary metric.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

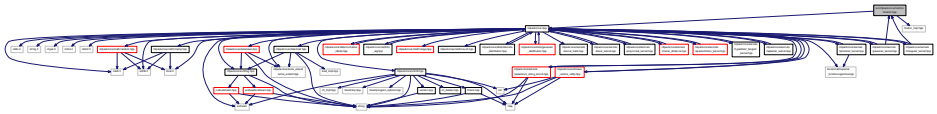
MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

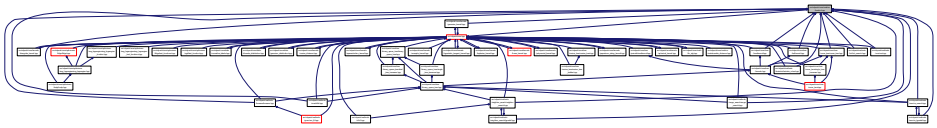
Definition in file **ip_metric.hpp**.

22.75 src/mlpack/core/metrics/lmetric.hpp File Reference

Include dependency graph for lmetric.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class **mlpack::metric::LMetric**< **Power, TakeRoot** >

  The L_p metric for arbitrary integer p, with an option to take the root.

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::metric**

Typedefs

- typedef LMetric< INT_MAX, false > **mlpack::metric::ChebyshevDistance**

- typedef LMetric< 2, true > **mlpack::metric::EuclideanDistance**
- typedef LMetric< 1, false > **mlpack::metric::ManhattanDistance**
- typedef LMetric< 2, false > **mlpack::metric::SquaredEuclideanDistance**

22.75.1 Detailed Description

Author

Ryan Curtin

Generalized L-metric, allowing both squared distances to be returned as well as non-squared distances. The squared distances are faster to compute.

This also gives several convenience typedefs for commonly used L-metrics.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

Definition in file **lmetric.hpp**.

22.76 src/mlpack/core/metrics/mahalanobis_distance.hpp File Reference

Include dependency graph for mahalanobis_distance.hpp:

Classes

- class **mlpack::metric::MahalanobisDistance**< **t_take_root** >

  The Mahalanobis distance, which is essentially a stretched Euclidean distance.

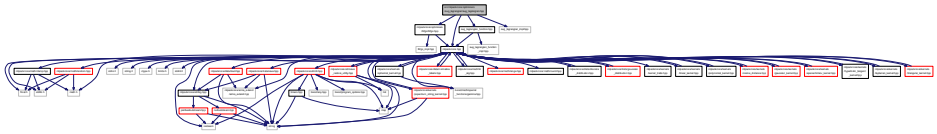
Namespaces

- **mlpack**

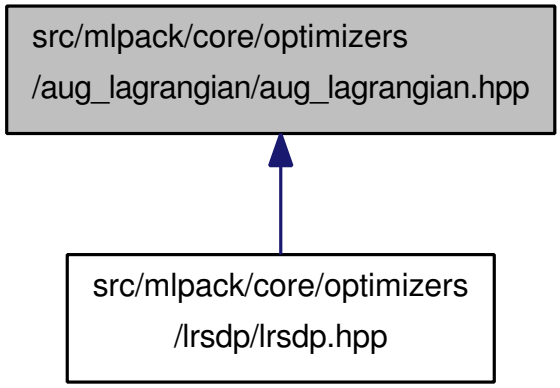
  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::metric**

22.77 src/mlpack/core/optimizers/aug_lagrangian/aug_lagrangian.hpp File Reference

Include dependency graph for aug_lagrangian.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class **mlpack::optimization::AugLagrangian< LagrangianFunction >**

  *The **AugLagrangian** (p. 363) class implements the Augmented Lagrangian method of optimization.*

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::optimization**

22.77.1 Detailed Description

Author

Ryan Curtin

Definition of AugLagrangian class, which implements the Augmented Lagrangian optimization method (also called the 'method of multipliers'. This class uses the L-BFGS optimizer.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

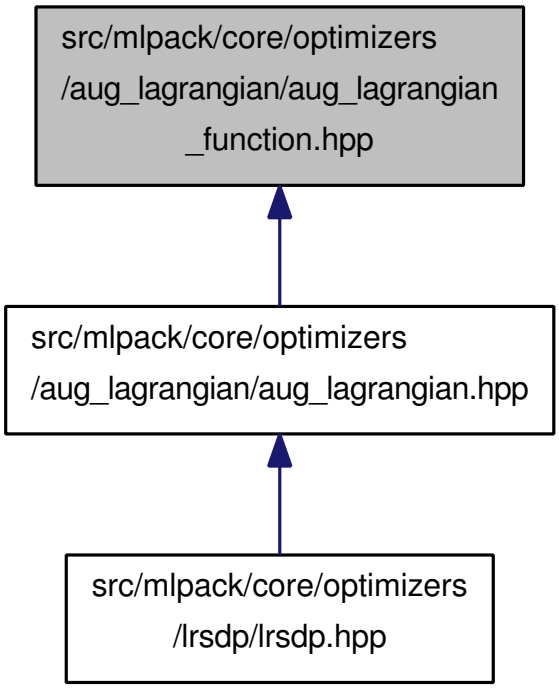
You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.-gnu.org/licenses/`.

Definition in file **aug_lagrangian.hpp**.

22.78 src/mlpack/core/optimizers/aug_lagrangian/aug_lagrangian_function.hpp File Reference

Include dependency graph for aug_lagrangian_function.hpp:

This graph shows which files directly or indirectly include this file:



Classes

- class **mlpack::optimization::AugLagrangianFunction< LagrangianFunction >**

  *This is a utility class used by **AugLagrangian** (p. 363), meant to wrap a LagrangianFunction into a function usable by a simple optimizer like L-BFGS.*

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.

- **mlpack::optimization**

22.78.1 Detailed Description

Author

Ryan Curtin

Contains a utility class for AugLagrangian.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

Definition in file **aug_lagrangian_function.hpp**.

22.79 src/mlpack/core/optimizers/aug_lagrangian/aug_lagrangian_test_functions.hpp File Reference

Include dependency graph for aug_lagrangian_test_functions.hpp:

Classes

- class **mlpack::optimization::AugLagrangianTestFunction**

  This function is taken from "Practical Mathematical Optimization" (Snyman), section 5.3.8 ("Application of the Augmented Lagrangian Method").

- class **mlpack::optimization::GockenbachFunction**

  This function is taken from M.

- class **mlpack::optimization::LovaszThetaSDP**

  This function is the Lovasz-Theta semidefinite program, as implemented in the following paper:

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.

- **mlpack::optimization**

22.79.1 Detailed Description

Author

Ryan Curtin

Define test functions for the augmented Lagrangian method.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

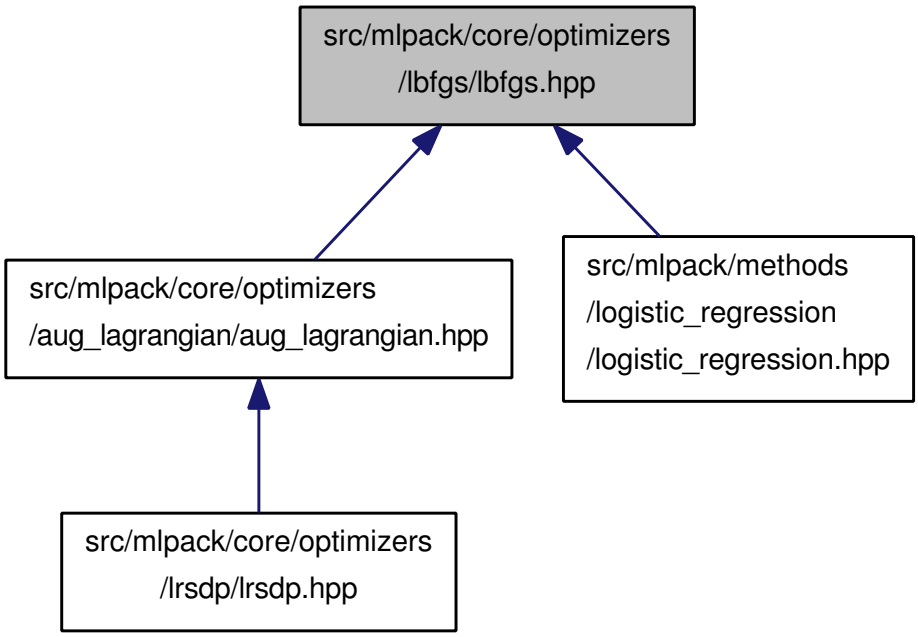
You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

Definition in file **aug_lagrangian_test_functions.hpp**.

22.80 src/mlpack/core/optimizers/lbfgs/lbfgs.hpp File Reference

Include dependency graph for lbfgs.hpp:

This graph shows which files directly or indirectly include this file:



Classes

- class **mlpack::optimization::L_BFGS**< **FunctionType** >

  The generic L-BFGS optimizer, which uses a back-tracking line search algorithm to minimize a function.

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::optimization**

22.80.1 Detailed Description

Author

Dongryeol Lee
Ryan Curtin

The generic L-BFGS optimizer.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

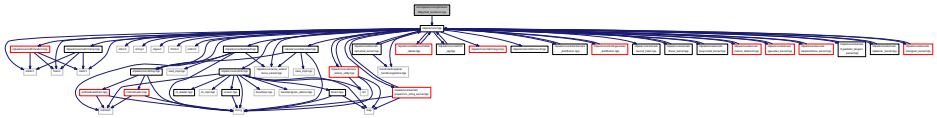
MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see http://www.gnu.org/licenses/.

Definition in file **lbfgs.hpp**.

22.81 src/mlpack/core/optimizers/lbfgs/test_functions.hpp File Reference

Include dependency graph for test_functions.hpp:



Classes

- class **mlpack::optimization::test::GeneralizedRosenbrockFunction**

  *The Generalized Rosenbrock function in n dimensions, defined by f(x) = sum_i^{n - 1} (f(i)(x)) f_i(x) = 100 * (x_i^2 - x_{i + 1})^2 + (1 - x_i)^2 x_0 = [-1.2, 1, -1.2, 1, ...].*
- class **mlpack::optimization::test::RosenbrockFunction**

  The Rosenbrock function, defined by f(x) = f1(x) + f2(x) f1(x) = 100 (x2 - x1^2)^2 f2(x) = (1 - x1)^2 x_0 = [-1.2, 1].
- class **mlpack::optimization::test::RosenbrockWoodFunction**

  The Generalized Rosenbrock function in 4 dimensions with the Wood Function in four dimensions.
- class **mlpack::optimization::test::WoodFunction**

  The Wood function, defined by f(x) = f1(x) + f2(x) + f3(x) + f4(x) + f5(x) + f6(x) f1(x) = 100 (x2 - x1^2)^2 f2(x) = (1 - x1)^2 f3(x) = 90 (x4 - x3^2)^2 f4(x) = (1 - x3)^2 f5(x) = 10 (x2 + x4 - 2)^2 f6(x) = (1 / 10) (x2 - x4)^2 x_0 = [-3, -1, -3, -1].

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::optimization**
- **mlpack::optimization::test**

22.81.1 Detailed Description

Author

Ryan Curtin

A collection of functions to test optimizers (in this case, L-BFGS). These come from the following paper:

"Testing Unconstrained Optimization Software" Jorge J. Moré, Burton S. Garbow, and Kenneth E. Hillstrom. 1981. ACM Trans. Math. Softw. 7, 1 (March 1981), 17-41. http://portal.acm.org/citation.cfm?id=355934.355936

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

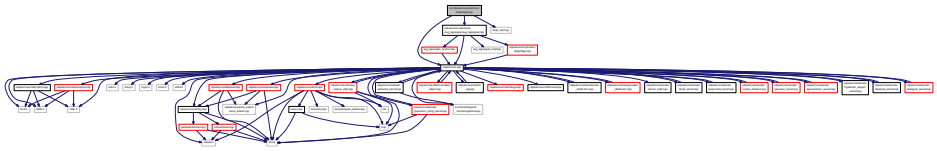
MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

Definition in file **test_functions.hpp**.

22.82 src/mlpack/core/optimizers/lrsdp/lrsdp.hpp File Reference

Include dependency graph for lrsdp.hpp:



Classes

- class **mlpack::optimization::LRSDP**

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::optimization**

22.82.1 Detailed Description

Author

Ryan Curtin

An implementation of Monteiro and Burer's formulation of low-rank semidefinite programs (LR-SDP).

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

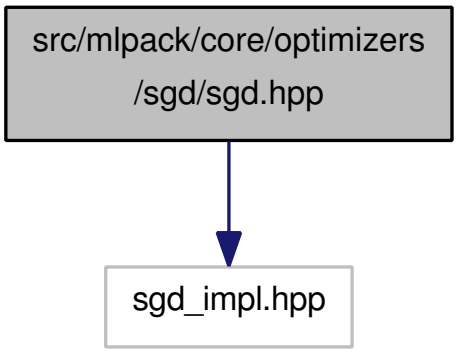
MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

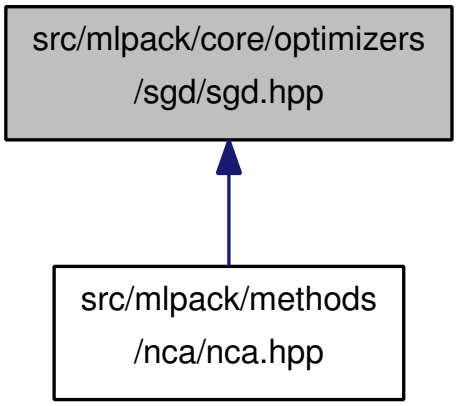
Definition in file **lrsdp.hpp**.

22.83 src/mlpack/core/optimizers/sgd/sgd.hpp File Reference

Include dependency graph for sgd.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class **mlpack::optimization::SGD**< **DecomposableFunctionType** >

  Stochastic Gradient Descent is a technique for minimizing a function which can be expressed as a sum of other functions.

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::optimization**

22.83.1 Detailed Description

Author

Ryan Curtin

Stochastic Gradient Descent (SGD).

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

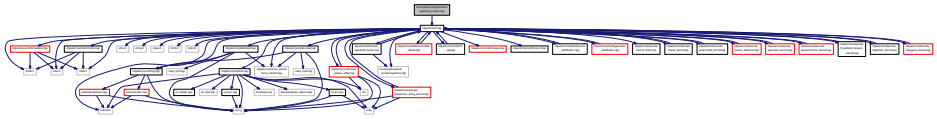
MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.-gnu.org/licenses/`.

Definition in file **sgd.hpp**.

22.84 src/mlpack/core/optimizers/sgd/test_function.hpp File Reference

Include dependency graph for test_function.hpp:



Classes

- class **mlpack::optimization::test::SGDTestFunction**

  Very, very simple test function which is the composite of three other functions.

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::optimization**
- **mlpack::optimization::test**

22.84.1 Detailed Description

Author

Ryan Curtin

Very simple test function for SGD.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

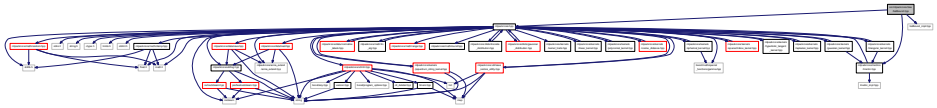
You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.-gnu.org/licenses/`.

Definition in file **test_function.hpp**.

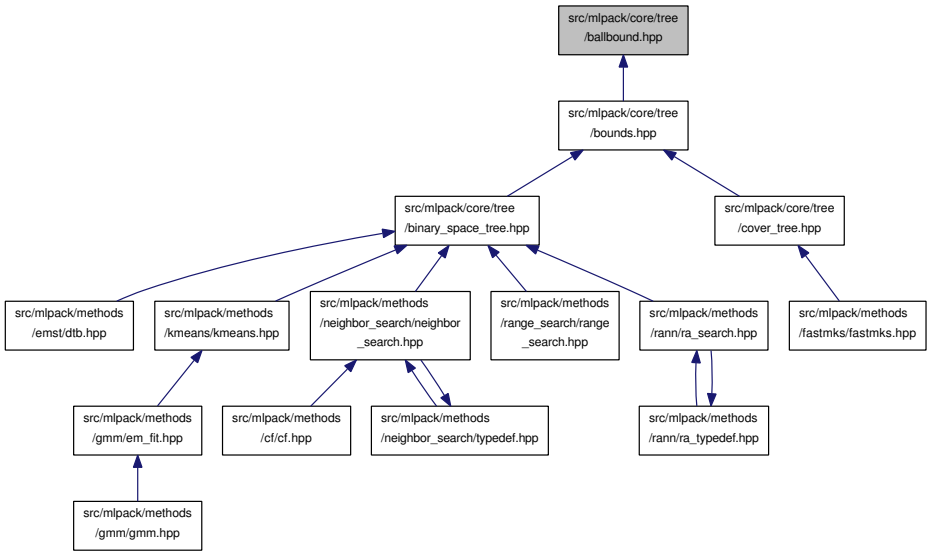
22.85 src/mlpack/core/tree/ballbound.hpp File Reference

Bounds that are useful for binary space partitioning trees.

Include dependency graph for ballbound.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class **mlpack::bound::BallBound**< **VecType** >

  Ball bound that works in the regular Euclidean metric space.

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.

- **mlpack::bound**

22.85.1 Detailed Description

Bounds that are useful for binary space partitioning trees. Interface to a ball bound that works in arbitrary metric spaces.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

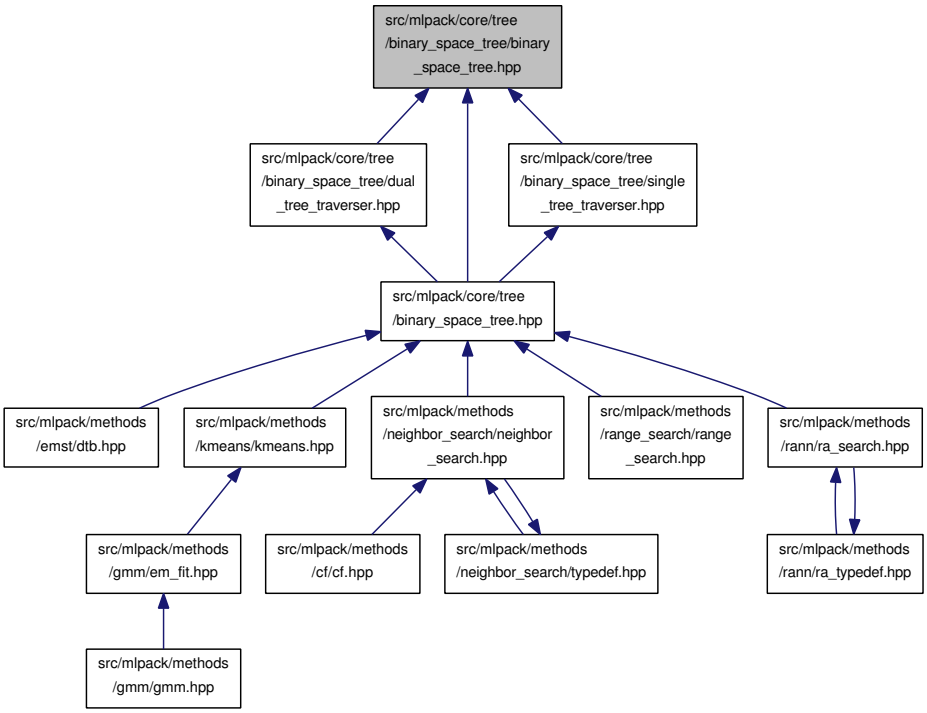
You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

Definition in file **ballbound.hpp**.

22.86 src/mlpack/core/tree/binary_space_tree/binary_space_tree.hpp File Reference

Include dependency graph for binary_space_tree.hpp:

This graph shows which files directly or indirectly include this file:



Classes

- class **mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >**

  A binary space partitioning tree, such as a KD-tree or a ball tree.

- class **mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::DualTreeTraverser< RuleType >**

  A dual-tree traverser for binary space trees; see dual_tree_traverser.hpp.

- class **mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::SingleTreeTraverser< RuleType >**

  A single-tree traverser for binary space trees; see single_tree_traverser.hpp for implementation.

Namespaces

- **mlpack**

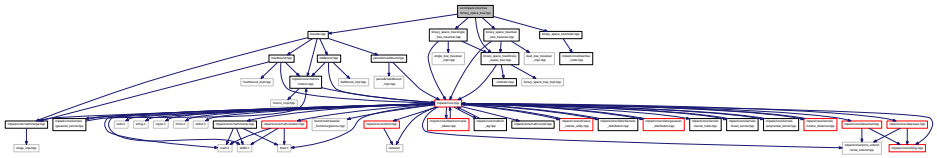
  Linear algebra utility functions, generally performed on matrices or vectors.

- **mlpack::tree**

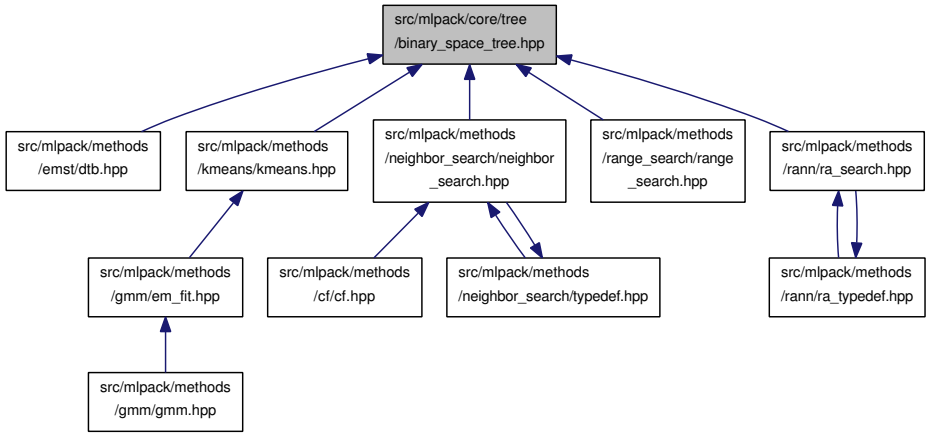
  Trees and tree-building procedures.

22.87 src/mlpack/core/tree/binary_space_tree.hpp File Reference

Include dependency graph for binary_space_tree.hpp:

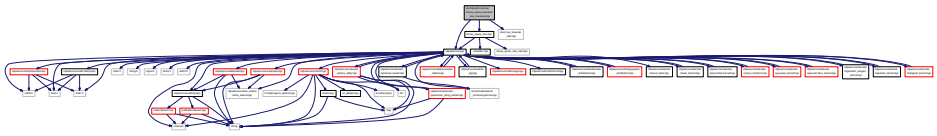


This graph shows which files directly or indirectly include this file:

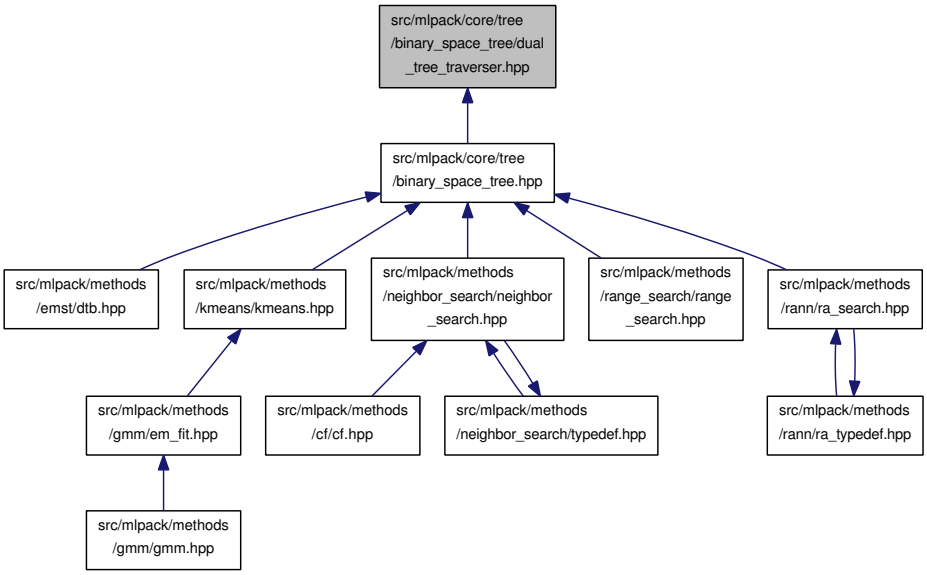


22.88 src/mlpack/core/tree/binary_space_tree/dual_tree_traverser.hpp File Reference

Include dependency graph for dual_tree_traverser.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class **mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::DualTreeTraverser< RuleType >**

  A dual-tree traverser for binary space trees; see dual_tree_traverser.hpp.

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.

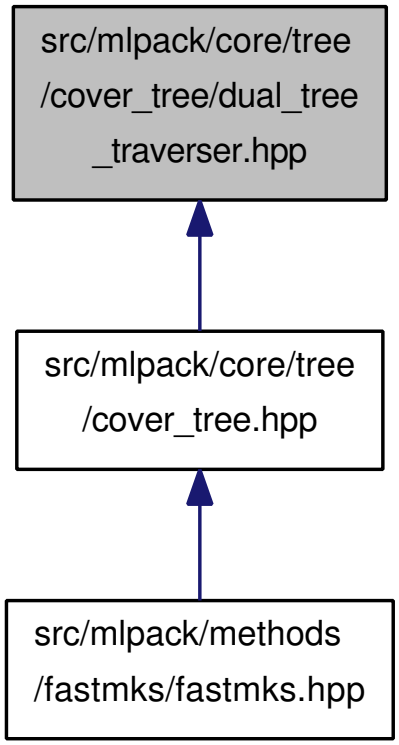
- **mlpack::tree**

  Trees and tree-building procedures.

22.89 src/mlpack/core/tree/cover_tree/dual_tree_traverser.hpp File Reference

Include dependency graph for dual_tree_traverser.hpp:

This graph shows which files directly or indirectly include this file:



Classes

- class **mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::DualTreeTraverser< RuleType >**

  A dual-tree cover tree traverser; see dual_tree_traverser.hpp.

- struct **mlpack::tree::DualCoverTreeMapEntry< MetricType, RootPointPolicy, StatisticType >**

  Forward declaration of struct to be used for traversal.

Namespaces

- **mlpack**

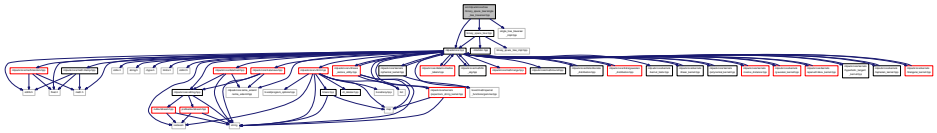
  Linear algebra utility functions, generally performed on matrices or vectors.

- **mlpack::tree**

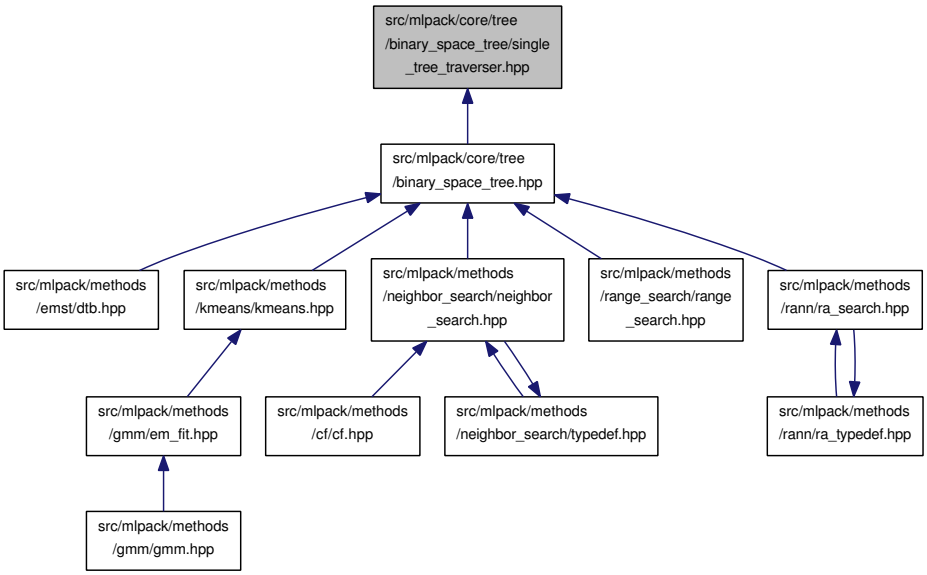
  Trees and tree-building procedures.

22.90 src/mlpack/core/tree/binary_space_tree/single_tree_traverser.hpp File Reference

Include dependency graph for single_tree_traverser.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class **mlpack::tree::BinarySpaceTree< BoundType, StatisticType, MatType >::SingleTreeTraverser< RuleType >**

  A single-tree traverser for binary space trees; see single_tree_traverser.hpp for implementation.

Namespaces

- **mlpack**

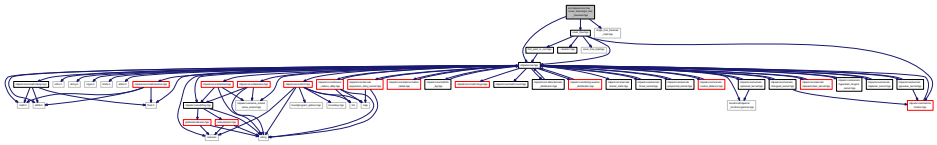
  Linear algebra utility functions, generally performed on matrices or vectors.

- **mlpack::tree**

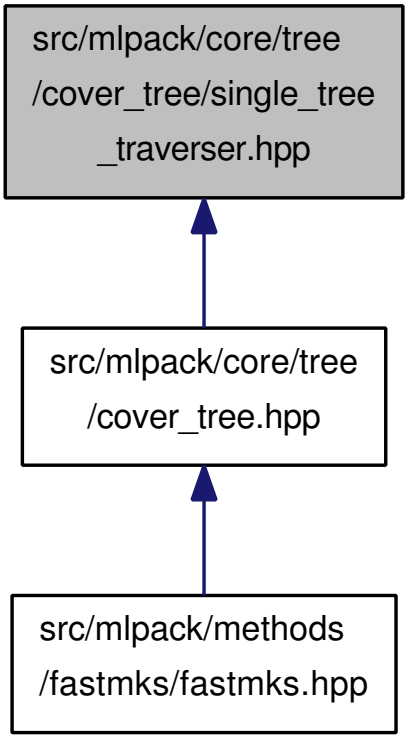
  Trees and tree-building procedures.

22.91 src/mlpack/core/tree/cover_tree/single_tree_traverser.hpp File Reference

Include dependency graph for single_tree_traverser.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class **mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::SingleTreeTraverser< RuleType >**

  A single-tree cover tree traverser; see single_tree_traverser.hpp for implementation.

Namespaces

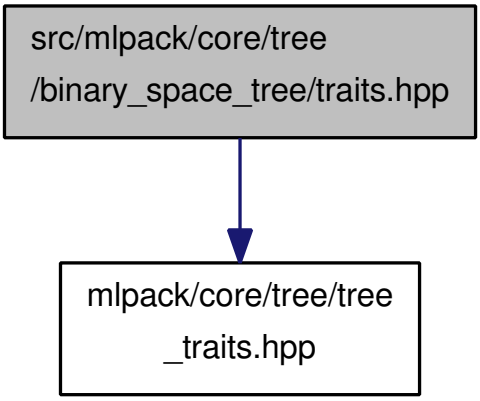
- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::tree**

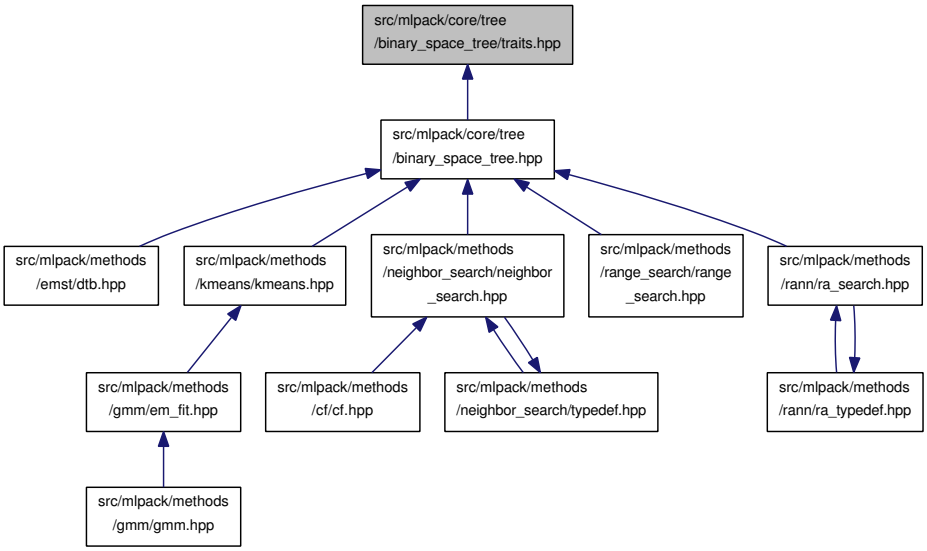
  Trees and tree-building procedures.

22.92 src/mlpack/core/tree/binary_space_tree/traits.hpp File Reference

Include dependency graph for traits.hpp:



This graph shows which files directly or indirectly include this file:



Classes

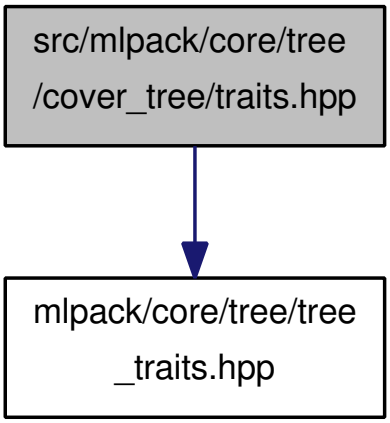
- class **mlpack::tree::TreeTraits< BinarySpaceTree< BoundType, StatisticType, MatType > >**
  *This is a specialization of the TreeType class to the **BinarySpaceTree** (p. 459) tree type.*

Namespaces

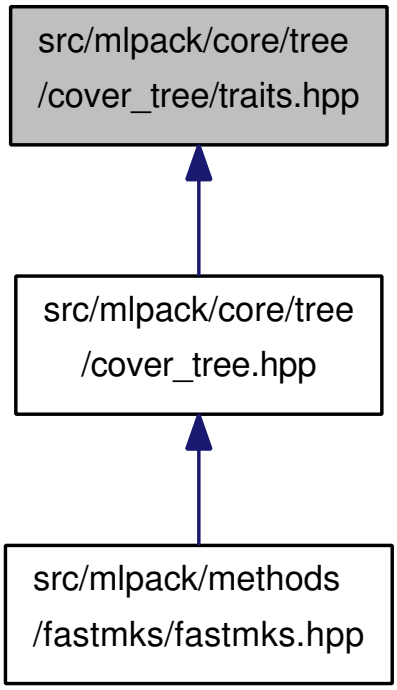
- **mlpack**
  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::tree**
  Trees and tree-building procedures.

22.93 src/mlpack/core/tree/cover_tree/traits.hpp File Reference

Include dependency graph for traits.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class **mlpack::tree::TreeTraits**< **CoverTree**< **MetricType, RootPointPolicy, StatisticType** > >

  *The specialization of the **TreeTraits*** (p. *521*) *class for the **CoverTree*** (p. *491*) *tree type.*

Namespaces

- **mlpack**

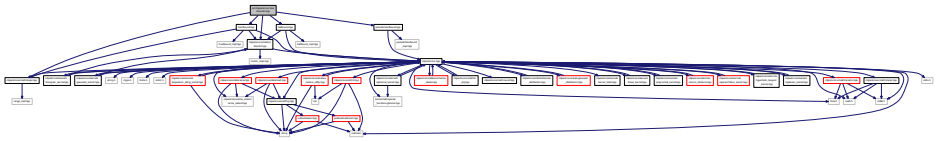
  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::tree**

  Trees and tree-building procedures.

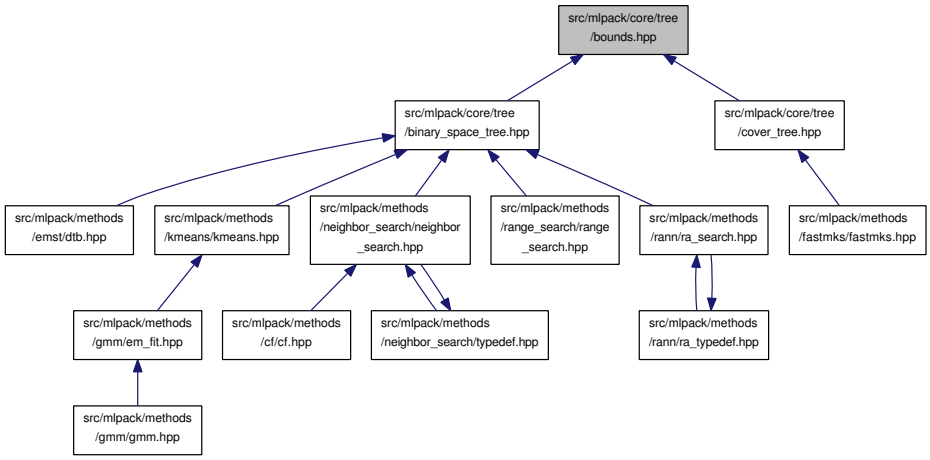
22.94 src/mlpack/core/tree/bounds.hpp File Reference

Bounds that are useful for binary space partitioning trees.

Include dependency graph for bounds.hpp:



This graph shows which files directly or indirectly include this file:



22.94.1 Detailed Description

Bounds that are useful for binary space partitioning trees. This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

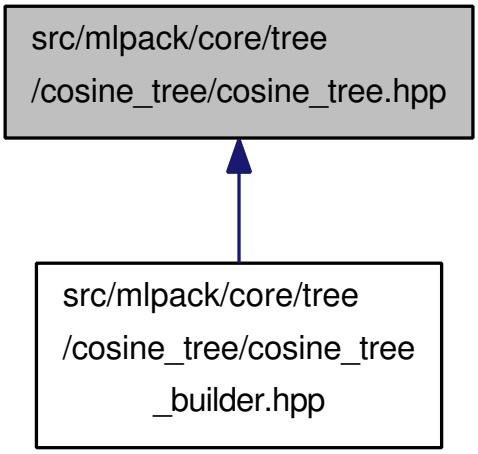
You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.-gnu.org/licenses/`.

Definition in file **bounds.hpp**.

22.95 src/mlpack/core/tree/cosine_tree/cosine_tree.hpp File Reference

Include dependency graph for cosine_tree.hpp:

This graph shows which files directly or indirectly include this file:



Classes

- class **mlpack::tree::CosineTree**

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::tree**

  Trees and tree-building procedures.

22.95.1 Detailed Description

Author

Mudit Raj Gupta

Definition of Cosine Tree.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

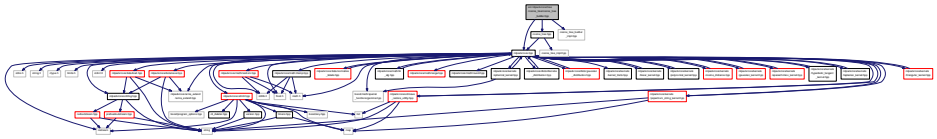
MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

Definition in file **cosine_tree.hpp**.

22.96 src/mlpack/core/tree/cosine_tree/cosine_tree_builder.hpp File Reference

Include dependency graph for cosine_tree_builder.hpp:



Classes

- class **mlpack::tree::CosineTreeBuilder**

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::tree**

  Trees and tree-building procedures.

22.96.1 Detailed Description

Author

Mudit Raj Gupta

Helper class to build the cosine tree

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

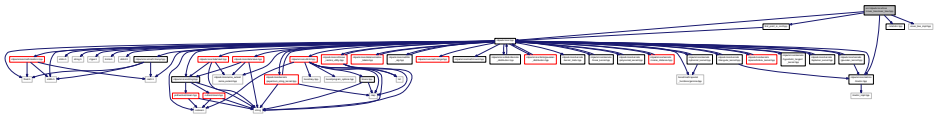
MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

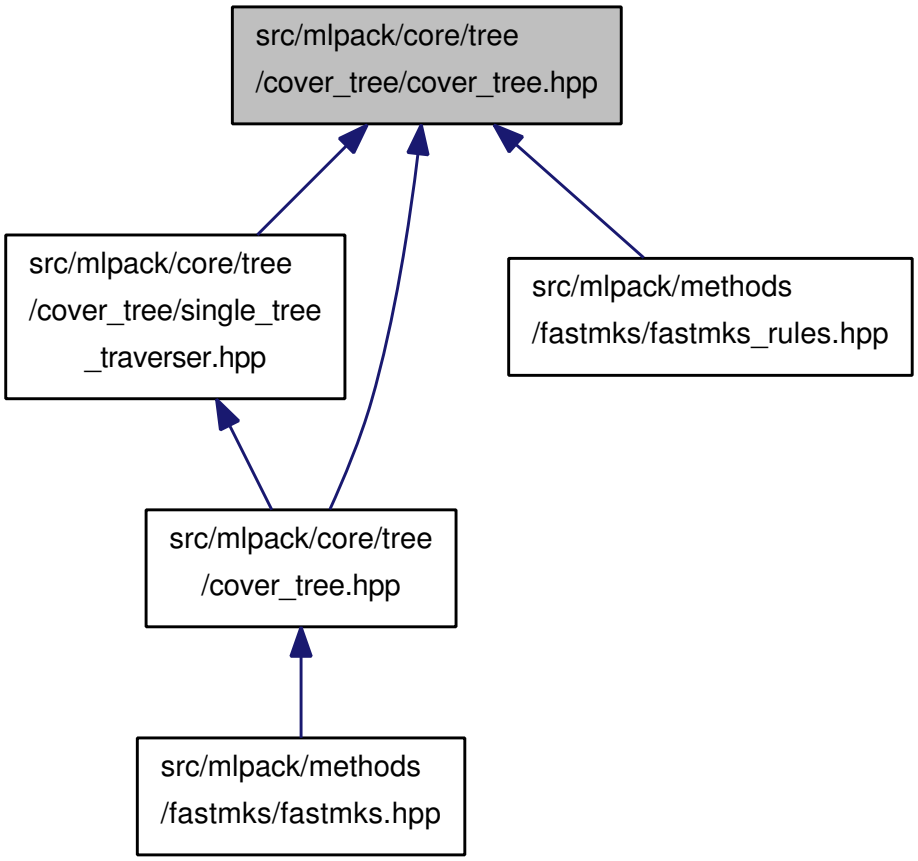
Definition in file **cosine_tree_builder.hpp**.

22.97 src/mlpack/core/tree/cover_tree/cover_tree.hpp File Reference

Include dependency graph for cover_tree.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class **mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >**

  A cover tree is a tree specifically designed to speed up nearest-neighbor computation in high-dimensional spaces.
- class **mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::DualTreeTraverser< RuleType >**

  A dual-tree cover tree traverser; see dual_tree_traverser.hpp.
- class **mlpack::tree::CoverTree< MetricType, RootPointPolicy, StatisticType >::SingleTreeTraverser< RuleType >**

  A single-tree cover tree traverser; see single_tree_traverser.hpp for implementation.

Namespaces

- **mlpack**

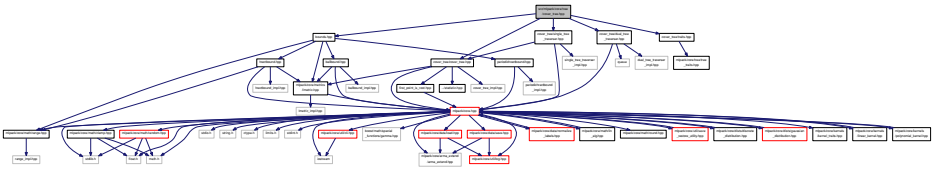
  Linear algebra utility functions, generally performed on matrices or vectors.

- **mlpack::tree**

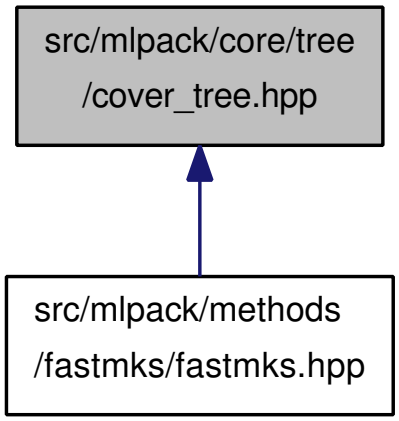
  Trees and tree-building procedures.

22.98 src/mlpack/core/tree/cover_tree.hpp File Reference

Include dependency graph for cover_tree.hpp:

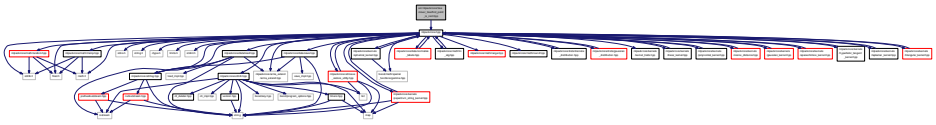


This graph shows which files directly or indirectly include this file:

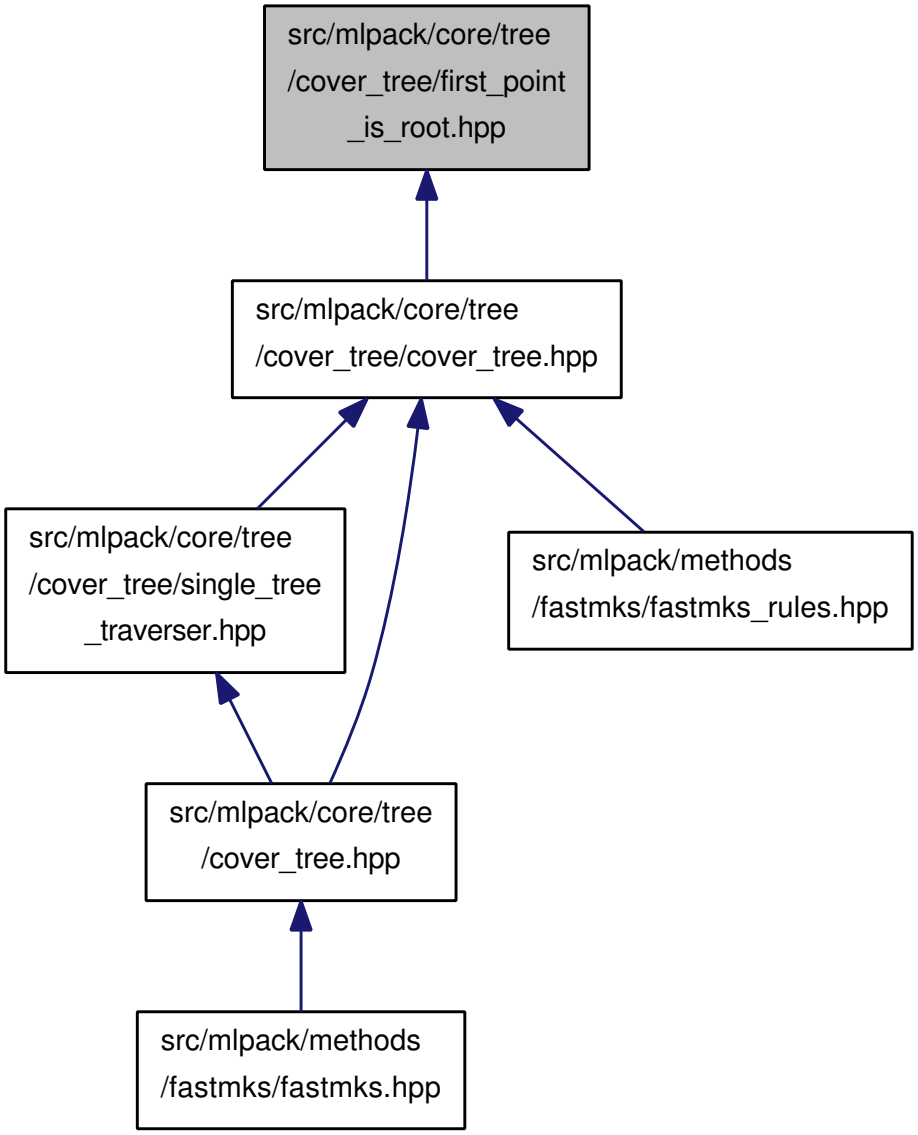


22.99 src/mlpack/core/tree/cover_tree/first_point_is_root.hpp File Reference

Include dependency graph for first_point_is_root.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class **mlpack::tree::FirstPointIsRoot**

  *This class is meant to be used as a choice for the policy class RootPointPolicy of the **CoverTree** (p. 491) class.*

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::tree**

  Trees and tree-building procedures.

22.99.1 Detailed Description

Author

Ryan Curtin

A very simple policy for the cover tree; the first point in the dataset is chosen as the root of the cover tree.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

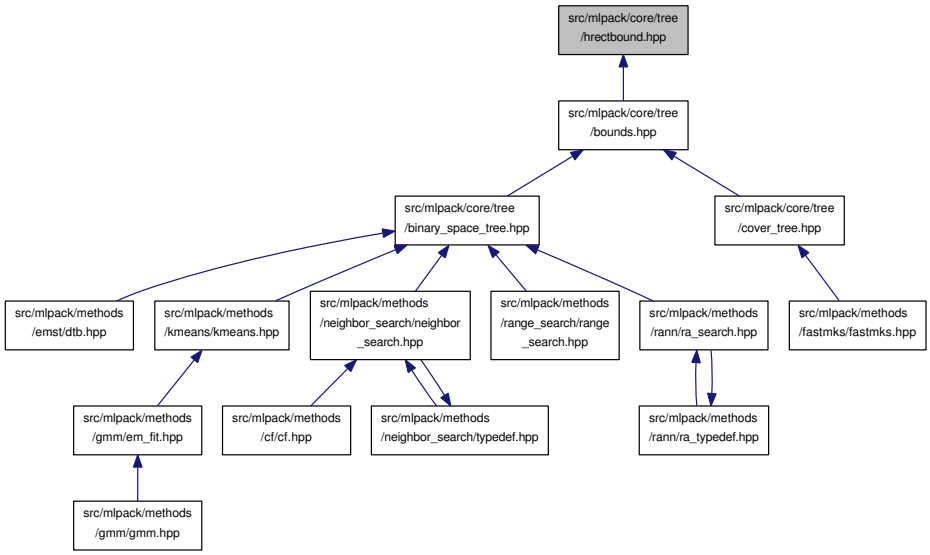
Definition in file **first_point_is_root.hpp**.

22.100 src/mlpack/core/tree/hrectbound.hpp File Reference

Bounds that are useful for binary space partitioning trees.

Include dependency graph for hrectbound.hpp:

This graph shows which files directly or indirectly include this file:



Classes

- class **mlpack::bound::HRectBound**< **Power, TakeRoot** >

  Hyper-rectangle bound for an L-metric.

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::bound**

22.100.1 Detailed Description

Bounds that are useful for binary space partitioning trees. This file describes the interface for the HRectBound class, which implements a hyperrectangle bound.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

Definition in file **hrectbound.hpp**.

22.101 src/mlpack/core/tree/mrkd_statistic.hpp File Reference

Include dependency graph for mrkd_statistic.hpp:

Classes

- class **mlpack::tree::MRKDStatistic**

  Statistic for multi-resolution kd-trees.

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::tree**

  Trees and tree-building procedures.

22.101.1 Detailed Description

Author

James Cline

Definition of the statistic for multi-resolution kd-trees.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

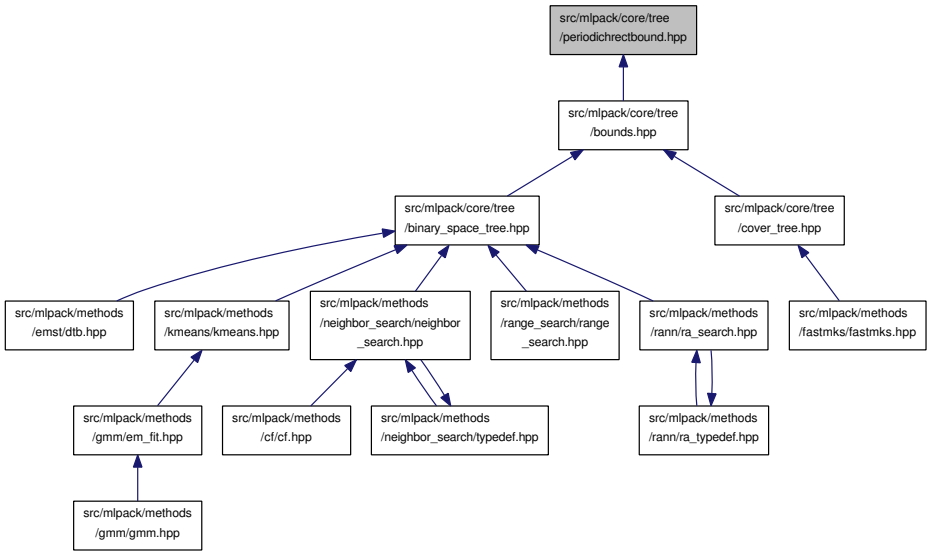
Definition in file **mrkd_statistic.hpp**.

22.102 src/mlpack/core/tree/periodichrectbound.hpp File Reference

Bounds that are useful for binary space partitioning trees.

Include dependency graph for periodichrectbound.hpp:

This graph shows which files directly or indirectly include this file:



Classes

- class **mlpack::bound::PeriodicHRectBound< t_pow >**

  Hyper-rectangle bound for an L-metric.

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.

- **mlpack::bound**

22.102.1 Detailed Description

Bounds that are useful for binary space partitioning trees. This file describes the interface for the PeriodicHRectBound policy, which implements a hyperrectangle bound in a periodic space.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

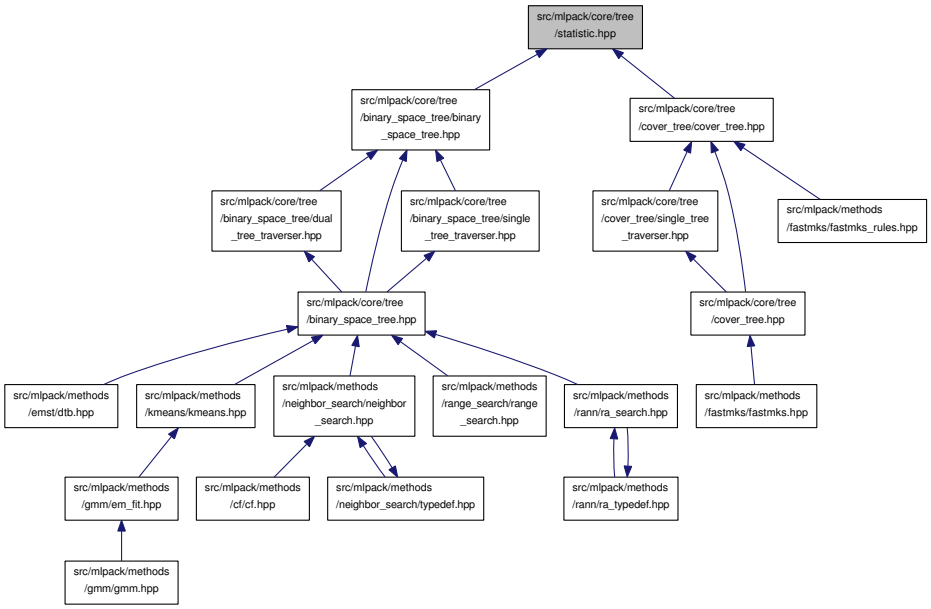
You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

Definition in file **periodichrectbound.hpp**.

22.103 src/mlpack/core/tree/statistic.hpp File Reference

Definition of the policy type for the statistic class.

This graph shows which files directly or indirectly include this file:



Classes

- class **mlpack::tree::EmptyStatistic**

  Empty statistic if you are not interested in storing statistics in your tree.

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::tree**

  Trees and tree-building procedures.

22.103.1 Detailed Description

Definition of the policy type for the statistic class. You should define your own statistic that looks like EmptyStatistic.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

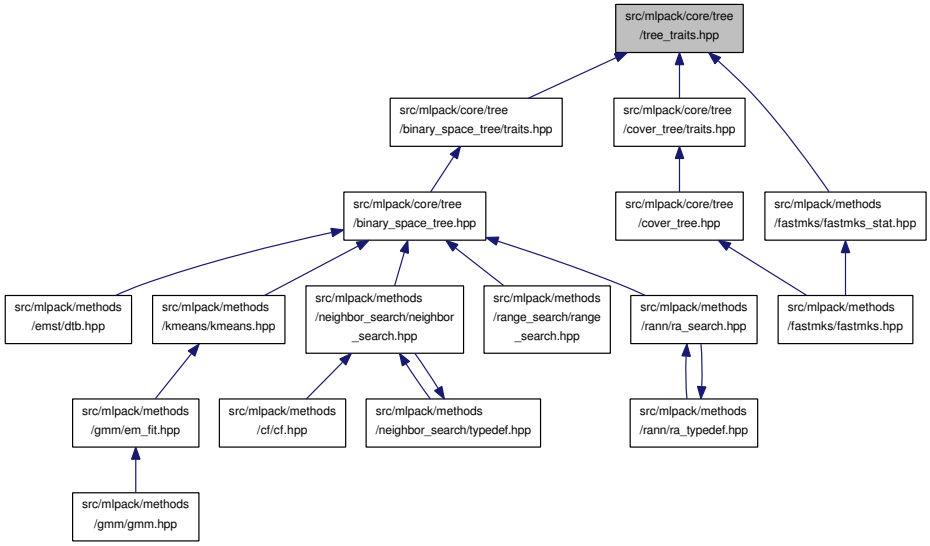
MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

Definition in file **statistic.hpp**.

22.104 src/mlpack/core/tree/tree_traits.hpp File Reference

This graph shows which files directly or indirectly include this file:



Classes

- class **mlpack::tree::TreeTraits< TreeType >**

  *The **TreeTraits** (p. 521) class provides compile-time information on the characteristics of a given tree type.*

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::tree**

  Trees and tree-building procedures.

22.104.1 Detailed Description

Author

Ryan Curtin

This file implements the basic, unspecialized TreeTraits class, which provides information about tree types. If you create a tree class, you should specialize this class with the characteristics of your tree.

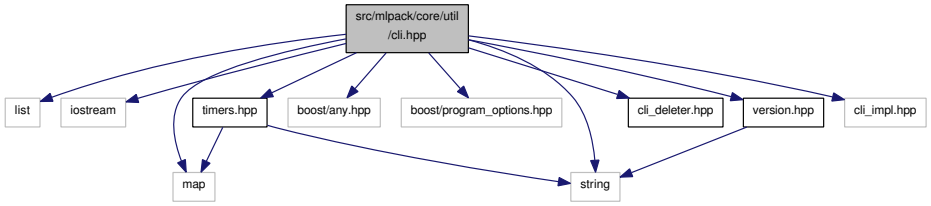
This file is part of MLPACK 1.0.8.

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

Definition in file **tree_traits.hpp**.

22.105 src/mlpack/core/util/cli.hpp File Reference

Include dependency graph for cli.hpp:



This graph shows which files directly or indirectly include this file:

Classes

- class **mlpack::CLI**

  Parses the command line for parameters and holds user-specified parameters.
- struct **mlpack::ParamData**

  *Aids in the extensibility of **CLI** (p. 132) by focusing potential changes into one structure.*

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::util**

Macros

- #define **PARAM**(T, ID, DESC, ALIAS, DEF, REQ)

  Define an input parameter.
- #define **PARAM_DOUBLE**(ID, DESC, ALIAS, DEF) **PARAM**(double, ID, DESC, ALIAS, DEF, false)

  Define a double parameter.
- #define **PARAM_DOUBLE_REQ**(ID, DESC, ALIAS)

  Define a required double parameter.
- #define **PARAM_FLAG**(ID, DESC, ALIAS) PARAM_FLAG_INTERNAL(ID, DESC, ALIAS);

  Define a flag parameter.
- #define **PARAM_FLOAT**(ID, DESC, ALIAS, DEF) **PARAM**(float, ID, DESC, ALIAS, DEF, false)

Define a floating-point parameter.

- #define **PARAM_FLOAT_REQ**(ID, DESC, ALIAS)

  Define a required floating-point parameter.
- #define **PARAM_INT**(ID, DESC, ALIAS, DEF) **PARAM**(int, ID, DESC, ALIAS, DEF, false)

  Define an integer parameter.
- #define **PARAM_INT_REQ**(ID, DESC, ALIAS) **PARAM**(int, ID, DESC, ALIAS, 0, true)

  Define a required integer parameter.
- #define **PARAM_STRING**(ID, DESC, ALIAS, DEF) **PARAM**(std::string, ID, DESC, ALIAS, DEF, false)

  Define a string parameter.
- #define **PARAM_STRING_REQ**(ID, DESC, ALIAS)

  Define a required string parameter.
- #define **PARAM_VECTOR**(T, ID, DESC, ALIAS) **PARAM**(std::vector<T>, ID, DESC, ALIAS, std::vector<T>(), false)

  Define a vector parameter.
- #define **PARAM_VECTOR_REQ**(T, ID, DESC, ALIAS)

  Define a required vector parameter.
- #define **PROGRAM_INFO**(NAME, DESC)

  Document an executable.
- #define **TYPENAME**(x) (std::string(typeid(x).name()))

  The TYPENAME macro is used internally to convert a type into a string.

22.105.1 Detailed Description

Author

Matthew Amidon

This file implements the CLI subsystem which is intended to replace FX. This can be used more or less regardless of context. In the future, it might be expanded to include file I/O.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

Definition in file **cli.hpp**.

22.105.2 Macro Definition Documentation

22.105.2.1 #define PARAM(*T, ID, DESC, ALIAS, DEF, REQ*)

Value:

```
static mlpack::util::Option<T>        JOIN(JOIN(io_option_dummy_object_, __LINE__), opt) (false, DEF, ID, \
      DESC, ALIAS, REQ);
```

Define an input parameter.

Don't use this function; use the other ones above that call it. Note that we are using the **LINE** macro for naming these actual parameters when **COUNTER** does not exist, which is a bit of an ugly hack... but this is the preprocessor, after all. We don't have much choice other than ugliness.

Parameters

T	Type of the parameter.
ID	Name of the parameter.
DESC	Description of the parameter (1-2 sentences).
ALIAS	Alias for this parameter (one letter).
DEF	Default value of the parameter.
REQ	Whether or not parameter is required (boolean value).

Definition at line 357 of file cli.hpp.

22.105.2.2 #define PARAM_DOUBLE(*ID, DESC, ALIAS, DEF*) PARAM(double, ID, DESC, ALIAS, DEF, false)

Define a double parameter.

The parameter can then be specified on the command line with –ID=value.

Parameters

ID	Name of the parameter.
DESC	Quick description of the parameter (1-2 sentences).
ALIAS	An alias for the parameter (one letter).
DEF	Default value of the parameter.

See Also

mlpack::CLI (p. 132), **PROGRAM_INFO()** (p. 630)

Bug The **COUNTER** variable is used in most cases to guarantee a unique global identifier for options declared using the PARAM_∗() macros. However, not all compilers have this support–most notably, gcc $<$ 4.3. In that case, the **LINE** macro is used as an attempt to get a unique global identifier, but collisions are still possible, and they produce bizarre error messages. See `http://mlpack.org/trac/ticket/74` for more information.

Definition at line 146 of file cli.hpp.

22.105.2.3 #define PARAM_DOUBLE_REQ(*ID, DESC, ALIAS*)

Value:

```
PARAM(double, ID, DESC, ALIAS, \
    0.0f, true)
```

Define a required double parameter.

The parameter must then be specified on the command line with –ID=value.

Parameters

ID	Name of the parameter.
DESC	Quick description of the parameter (1-2 sentences).
ALIAS	An alias for the parameter (one letter).

See Also

mlpack::CLI (p. 132), **PROGRAM_INFO()** (p. 630)

Bug The **COUNTER** variable is used in most cases to guarantee a unique global identifier for options declared using the PARAM_∗() macros. However, not all compilers have this support–most notably, gcc $<$ 4.3. In that case, the **LINE** macro is used as an attempt to get a unique global identifier, but collisions are still possible, and they produce bizarre error messages. See `http://mlpack.org/trac/ticket/74` for more information.

Definition at line 267 of file cli.hpp.

22.105.2.4 #define PARAM_FLAG(*ID, DESC, ALIAS*) PARAM_FLAG_INTERNAL(ID, DESC, ALIAS);

Define a flag parameter.

Parameters

ID	Name of the parameter.
DESC	Quick description of the parameter (1-2 sentences).
ALIAS	An alias for the parameter (one letter).

See Also

mlpack::CLI (p. 132), **PROGRAM_INFO()** (p. 630)

Bug The **COUNTER** variable is used in most cases to guarantee a unique global identifier for options declared using the PARAM_∗() macros. However, not all compilers have this support–most notably, gcc $<$ 4.3. In that case, the **LINE** macro is used as an attempt to get a unique global identifier, but collisions are still possible, and they produce bizarre error messages. See `http://mlpack.org/trac/ticket/74` for more information.

Definition at line 74 of file cli.hpp.

22.105.2.5 #define PARAM_FLOAT(*ID, DESC, ALIAS, DEF*) PARAM(float, ID, DESC, ALIAS, DEF, false)

Define a floating-point parameter.

You should use PARAM_DOUBLE instead.

The parameter can then be specified on the command line with –ID=value.

Parameters

ID	Name of the parameter.
DESC	Quick description of the parameter (1-2 sentences).

ALIAS	An alias for the parameter (one letter).
DEF	Default value of the parameter.

See Also

mlpack::CLI (p. 132), **PROGRAM_INFO()** (p. 630)

Bug The **COUNTER** variable is used in most cases to guarantee a unique global identifier for options declared using the PARAM_*() macros. However, not all compilers have this support–most notably, gcc < 4.3. In that case, the **LINE** macro is used as an attempt to get a unique global identifier, but collisions are still possible, and they produce bizarre error messages. See `http://mlpack.org/trac/ticket/74` for more information.

Definition at line 122 of file cli.hpp.

22.105.2.6 #define PARAM_FLOAT_REQ(*ID, DESC, ALIAS*)

Value:

```
PARAM(float, ID, DESC, ALIAS, 0.0f, \
    true)
```

Define a required floating-point parameter.

You should probably use a double instead.

The parameter must then be specified on the command line with –ID=value. If ALIAS is equal to DEF_MOD (which is set using the **PROGRAM_INFO()** (p. 630) macro), the parameter can be specified with just –ID=value.

Parameters

ID	Name of the parameter.
DESC	Quick description of the parameter (1-2 sentences).
ALIAS	An alias for the parameter (one letter).

See Also

mlpack::CLI (p. 132), **PROGRAM_INFO()** (p. 630)

Bug The **COUNTER** variable is used in most cases to guarantee a unique global identifier for options declared using the PARAM_*() macros. However, not all compilers have this support–most notably, gcc < 4.3. In that case, the **LINE** macro is used as an attempt to get a unique global identifier, but collisions are still possible, and they produce bizarre error messages. See `http://mlpack.org/trac/ticket/74` for more information.

Definition at line 244 of file cli.hpp.

22.105.2.7 #define PARAM_INT(*ID, DESC, ALIAS, DEF*) PARAM(int, ID, DESC, ALIAS, DEF, false)

Define an integer parameter.

The parameter can then be specified on the command line with –ID=value.

Parameters

ID	Name of the parameter.
DESC	Quick description of the parameter (1-2 sentences).
ALIAS	An alias for the parameter (one letter).
DEF	Default value of the parameter.

See Also

mlpack::CLI (p. 132), **PROGRAM_INFO()** (p. 630)

Bug The **COUNTER** variable is used in most cases to guarantee a unique global identifier for options declared using the PARAM_∗() macros. However, not all compilers have this support–most notably, gcc < 4.3. In that case, the **LINE** macro is used as an attempt to get a unique global identifier, but collisions are still possible, and they produce bizarre error messages. See `http://mlpack.org/trac/ticket/74` for more information.

Definition at line 98 of file cli.hpp.

22.105.2.8 #define PARAM_INT_REQ(*ID, DESC, ALIAS*) PARAM(int, ID, DESC, ALIAS, 0, true)

Define a required integer parameter.

The parameter must then be specified on the command line with –ID=value.

Parameters

ID	Name of the parameter.
DESC	Quick description of the parameter (1-2 sentences).
ALIAS	An alias for the parameter (one letter).

See Also

mlpack::CLI (p. 132), **PROGRAM_INFO()** (p. 630)

Bug The **COUNTER** variable is used in most cases to guarantee a unique global identifier for options declared using the PARAM_∗() macros. However, not all compilers have this support–most notably, gcc < 4.3. In that case, the **LINE** macro is used as an attempt to get a unique global identifier, but collisions are still possible, and they produce bizarre error messages. See `http://mlpack.org/trac/ticket/74` for more information.

Definition at line 220 of file cli.hpp.

22.105.2.9 #define PARAM_STRING(*ID, DESC, ALIAS, DEF*) PARAM(std::string, ID, DESC, ALIAS, DEF, false)

Define a string parameter.

The parameter can then be specified on the command line with –ID=value. If ALIAS is equal to DEF_MOD (which is set using the **PROGRAM_INFO()** (p. 630) macro), the parameter can be specified with just –ID=value.

Parameters

ID	Name of the parameter.
DESC	Quick description of the parameter (1-2 sentences).
ALIAS	An alias for the parameter (one letter).
DEF	Default value of the parameter.

See Also

mlpack::CLI (p. 132), **PROGRAM_INFO()** (p. 630)

Bug The **COUNTER** variable is used in most cases to guarantee a unique global identifier for options declared using the PARAM_∗() macros. However, not all compilers have this support–most notably, gcc < 4.3. In that case, the **LINE** macro is used as an attempt to get a unique global identifier, but collisions are still possible, and they produce bizarre error messages. See `http://mlpack.org/trac/ticket/74` for more information.

Definition at line 171 of file cli.hpp.

22.105.2.10 #define PARAM_STRING_REQ(*ID, DESC, ALIAS*)

Value:

```
PARAM(std::string, ID, DESC, \
    ALIAS, "", true);
```

Define a required string parameter.

The parameter must then be specified on the command line with –ID=value.

Parameters

ID	Name of the parameter.
DESC	Quick description of the parameter (1-2 sentences).
ALIAS	An alias for the parameter (one letter).

See Also

mlpack::CLI (p. 132), **PROGRAM_INFO()** (p. 630)

Bug The **COUNTER** variable is used in most cases to guarantee a unique global identifier for options declared using the PARAM_∗() macros. However, not all compilers have this support–most notably, gcc < 4.3. In that case, the **LINE** macro is used as an attempt to get a unique global identifier, but collisions are still possible, and they produce bizarre error messages. See `http://mlpack.org/trac/ticket/74` for more information.

Definition at line 290 of file cli.hpp.

22.105.2.11 #define PARAM_VECTOR(*T, ID, DESC, ALIAS*) PARAM(std::vector<T>, ID, DESC, ALIAS, std::vector<T>(), false)

Define a vector parameter.

The parameter can then be specified on the command line with –ID=value.

Parameters

ID	Name of the parameter.
DESC	Quick description of the parameter (1-2 sentences).
ALIAS	An alias for the parameter (one letter).
DEF	Default value of the parameter.

See Also

mlpack::CLI (p. 132), **PROGRAM_INFO()** (p. 630)

Bug The **COUNTER** variable is used in most cases to guarantee a unique global identifier for options declared using the PARAM_∗() macros. However, not all compilers have this support–most notably, gcc $<$ 4.3. In that case, the **LINE** macro is used as an attempt to get a unique global identifier, but collisions are still possible, and they produce bizarre error messages. See `http://mlpack.org/trac/ticket/74` for more information.

Definition at line 195 of file cli.hpp.

22.105.2.12 #define PARAM_VECTOR_REQ(*T, ID, DESC, ALIAS*)

Value:

```
PARAM(std::vector<T>, ID, DESC, \
    ALIAS, std::vector<T>(), true);
```

Define a required vector parameter.

The parameter must then be specified on the command line with –ID=value.

Parameters

ID	Name of the parameter.
DESC	Quick description of the parameter (1-2 sentences).
ALIAS	An alias for the parameter (one letter).

See Also

mlpack::CLI (p. 132), **PROGRAM_INFO()** (p. 630)

Bug The **COUNTER** variable is used in most cases to guarantee a unique global identifier for options declared using the PARAM_∗() macros. However, not all compilers have this support–most notably, gcc $<$ 4.3. In that case, the **LINE** macro is used as an attempt to get a unique global identifier, but collisions are still possible, and they produce bizarre error messages. See `http://mlpack.org/trac/ticket/74` for more information.

Definition at line 313 of file cli.hpp.

22.105.2.13 #define PROGRAM_INFO(*NAME, DESC*)

Value:

```
static mlpack::util::ProgramDoc \
    io_programdoc_dummy_object = mlpack::util::ProgramDoc(NAME, DESC);
```

Document an executable.

Only one instance of this macro should be present in your program! Therefore, use it in the main.cpp (or corresponding executable) in your program.

See Also

mlpack::CLI (p. 132), **PARAM_FLAG()** (p. 626), **PARAM_INT()** (p. 627), **PARAM_DOUBLE()** (p. 625), **PARAM_STRING()** (p. 628), **PARAM_VECTOR()** (p. 629), **PARAM_INT_REQ()** (p. 628), **PARAM_DOUBLE_REQ()** (p. 625), **PARAM_STRING_REQ()** (p. 629), **PARAM_VECTOR_REQ()** (p. 630).

Parameters

NAME	Short string representing the name of the program.
DESC	Long string describing what the program does and possibly a simple usage example. Newlines should not be used here; this is taken care of by CLI (however, you can explicitly specify newlines to denote new paragraphs).

Definition at line 54 of file cli.hpp.

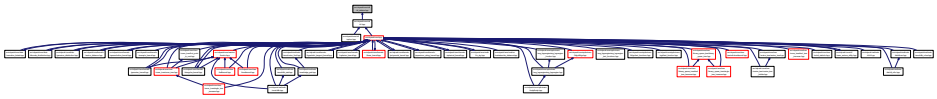
22.105.2.14 #define TYPENAME(*x*) (std::string(typeid(x).name()))

The TYPENAME macro is used internally to convert a type into a string.

Definition at line 372 of file cli.hpp.

22.106 src/mlpack/core/util/cli_deleter.hpp File Reference

This graph shows which files directly or indirectly include this file:



Classes

- class **mlpack::util::CLIDeleter**

  *Extremely simple class whose only job is to delete the existing **CLI** (p. 132) object at the end of execution.*

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::util**

Variables

- static CLIDeleter **mlpack::util::cliDeleter**

  Declare the deleter.

22.106.1 Detailed Description

Author

Ryan Curtin

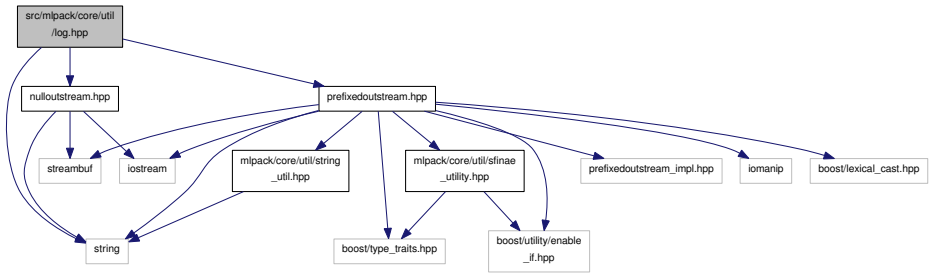
Definition of the CLIDeleter() class.

This file is part of MLPACK 1.0.8.

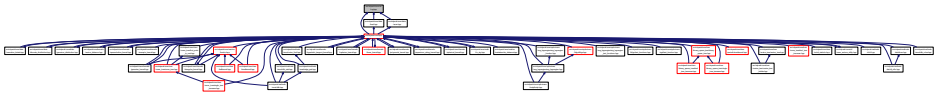
Definition in file **cli_deleter.hpp**.

22.107 src/mlpack/core/util/log.hpp File Reference

Include dependency graph for log.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class **mlpack::Log**

  Provides a convenient way to give formatted output.

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.

22.107.1 Detailed Description

Author

Matthew Amidon

Definition of the Log class.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

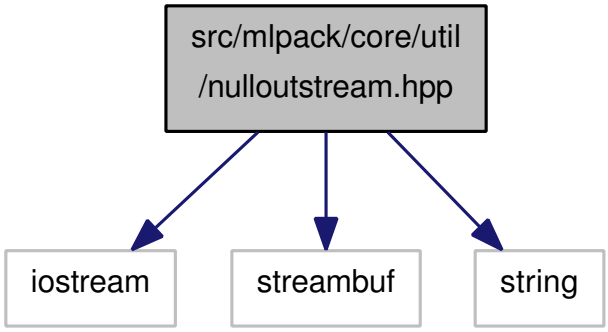
MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

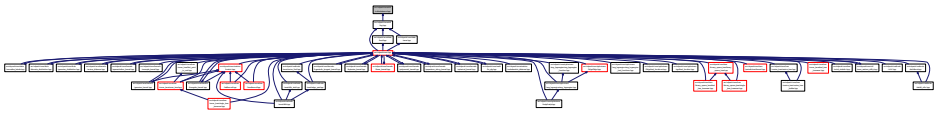
Definition in file **log.hpp**.

22.108 src/mlpack/core/util/nulloutstream.hpp File Reference

Include dependency graph for nulloutstream.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class **mlpack::util::NullOutStream**

  *Used for **Log::Debug** (p. 280) when not compiled with debugging symbols.*

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::util**

22.108.1 Detailed Description

Author

Ryan Curtin
Matthew Amidon

Definition of the NullOutStream class.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

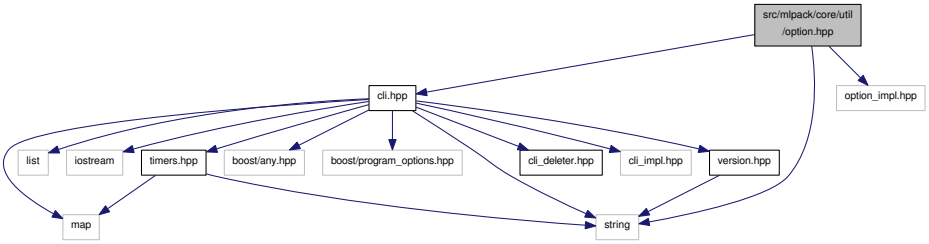
MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

Definition in file **nulloutstream.hpp**.

22.109 src/mlpack/core/util/option.hpp File Reference

Include dependency graph for option.hpp:



Classes

- class **mlpack::util::Option**< **N** >

  *A static object whose constructor registers a parameter with the **CLI** (p. 132) class.*
- class **mlpack::util::ProgramDoc**

  *A static object whose constructor registers program documentation with the **CLI** (p. 132) class.*

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::util**

22.109.1 Detailed Description

Author

Matthew Amidon

Definition of the Option class, which is used to define parameters which are used by CLI. The ProgramDoc class also resides here.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

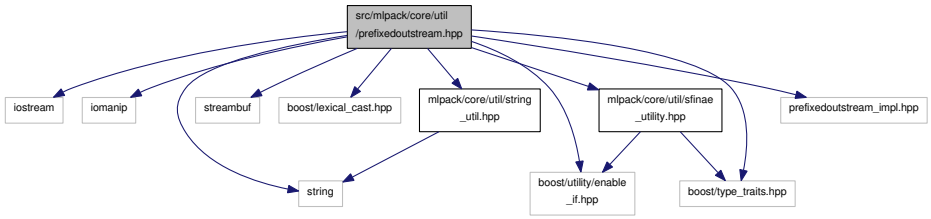
MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

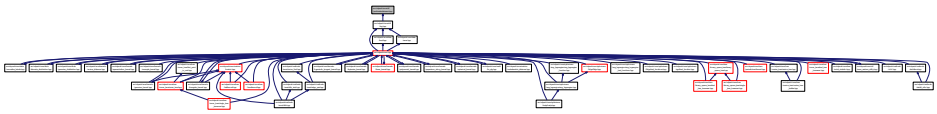
Definition in file **option.hpp**.

22.110 src/mlpack/core/util/prefixedoutstream.hpp File Reference

Include dependency graph for prefixedoutstream.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class **mlpack::util::PrefixedOutStream**

Allows us to output to an ostream with a prefix at the beginning of each line, in the same way we would output to cout or cerr.

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::util**

22.110.1 Detailed Description

Author

Ryan Curtin
Matthew Amidon

Declaration of the PrefixedOutStream class.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

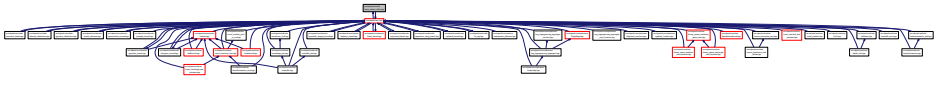
You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

Definition in file **prefixedoutstream.hpp**.

22.111 src/mlpack/core/util/save_restore_utility.hpp File Reference

Include dependency graph for save_restore_utility.hpp:

This graph shows which files directly or indirectly include this file:



Classes

- class **mlpack::util::SaveRestoreUtility**

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::util**

22.111.1 Detailed Description

Author

Neil Slagle

The SaveRestoreUtility provides helper functions in saving and restoring models. The current output file type is XML.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

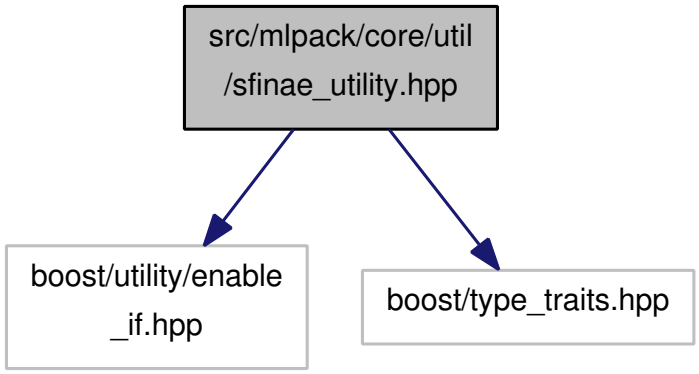
MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

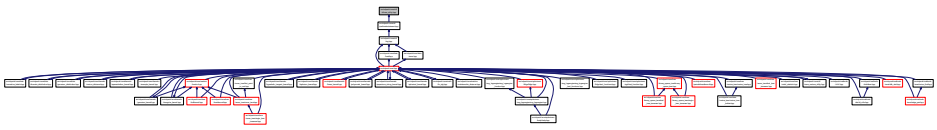
Definition in file **save_restore_utility.hpp**.

22.112 src/mlpack/core/util/sfinae_utility.hpp File Reference

Include dependency graph for sfinae_utility.hpp:



This graph shows which files directly or indirectly include this file:



Macros

- #define **HAS_MEM_FUNC**(FUNC, NAME)

22.112.1 Detailed Description

Author

Trironk Kiatkungwanglai

This file contains macro utilities for the SFINAE Paradigm. These utilities determine if classes passed in as template parameters contain members at compile time, which is useful for changing functionality depending on what operations an object is capable of performing.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

Definition in file **sfinae_utility.hpp**.

22.112.2 Macro Definition Documentation

22.112.2.1 #define HAS_MEM_FUNC(*FUNC, NAME*)

Value:

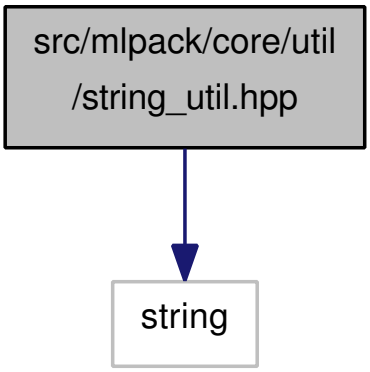
```
template<typename T, typename sig>                                          \
struct NAME {                                                               \
  typedef char yes[1];                                                      \
  typedef char no [2];                                                      \
  template<typename U, U> struct type_check;                                \
  template<typename _1> static yes &chk(type_check<sig, &_1::FUNC> *);      \
  template<typename   > static no  &chk(...);                               \
  static bool const value = sizeof(chk<T>(0)) == sizeof(yes);               \
};
```

Definition at line 50 of file sfinae_utility.hpp.

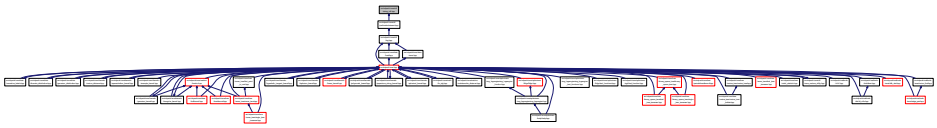
22.113 src/mlpack/core/util/string_util.hpp File Reference

Declares methods that are useful for writing formatting output.

Include dependency graph for string_util.hpp:



This graph shows which files directly or indirectly include this file:



Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::util**

Functions

- std::string **mlpack::util::Indent** (std::string input)

  A utility function that replaces all all newlines with a number of spaces depending on the indentation level.

22.113.1 Detailed Description

Declares methods that are useful for writing formatting output. This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

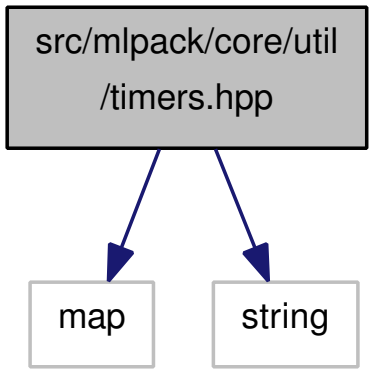
MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

Definition in file **string_util.hpp**.

22.114 src/mlpack/core/util/timers.hpp File Reference

Include dependency graph for timers.hpp:



This graph shows which files directly or indirectly include this file:

Classes

- class **mlpack::Timer**

  The timer class provides a way for MLPACK methods to be timed.
- class **mlpack::Timers**

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.

22.114.1 Detailed Description

Author

Matthew Amidon
Marcus Edel

Timers for MLPACK.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

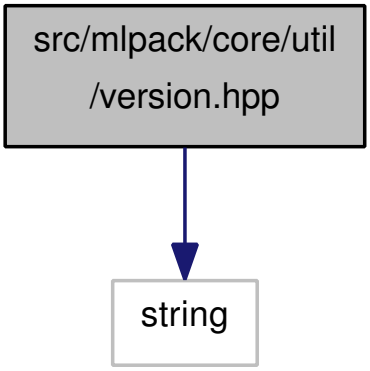
MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

Definition in file **timers.hpp**.

22.115 src/mlpack/core/util/version.hpp File Reference

Include dependency graph for version.hpp:



This graph shows which files directly or indirectly include this file:

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::util**

Macros

- #define **__MLPACK_VERSION_MAJOR** 1
- #define **__MLPACK_VERSION_MINOR** 0
- #define **__MLPACK_VERSION_PATCH** 8

Functions

- std::string **mlpack::util::GetVersion** ()

  This will return either "mlpack x.y.z" or "mlpack trunk-rXXXXX" depending on whether or not this is a stable version of mlpack or an svn revision.

22.115.1 Macro Definition Documentation

22.115.1.1 #define __MLPACK_VERSION_MAJOR 1

Definition at line 29 of file version.hpp.

22.115.1.2 #define __MLPACK_VERSION_MINOR 0

Definition at line 30 of file version.hpp.

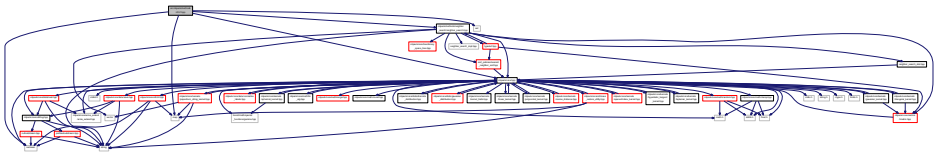
22.115.1.3 #define __MLPACK_VERSION_PATCH 8

Definition at line 31 of file version.hpp.

22.116 doc/guide/version.hpp File Reference

22.117 src/mlpack/methods/cf/cf.hpp File Reference

Include dependency graph for cf.hpp:



Classes

- class **mlpack::cf::CF**

  *This class implements Collaborative Filtering (**CF** (p. 126)).*

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::cf**

  Collaborative filtering.

22.117.1 Detailed Description

Author

Mudit Raj Gupta

Collaborative filtering.

Defines the CF class to perform collaborative filtering on the specified data set using alternating least squares (ALS).

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

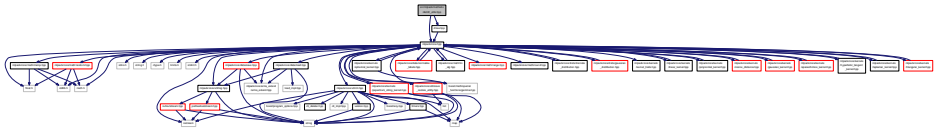
MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.-gnu.org/licenses/`.

Definition in file **cf.hpp**.

22.118 src/mlpack/methods/det/dt_utils.hpp File Reference

Include dependency graph for dt_utils.hpp:



Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.

- **mlpack::det**

  Density Estimation Trees.

Functions

- void **mlpack::det::PrintLeafMembership** (DTree ∗dtree, const arma::mat &data, const arma::Mat< size_t > &labels, const size_t numClasses, const std::string leafClassMembershipFile="")

  Print the membership of leaves of a density estimation tree given the labels and number of classes.

- void **mlpack::det::PrintVariableImportance** (const DTree ∗dtree, const std::string viFile="")

  Print the variable importance of each dimension of a density estimation tree.

- DTree ∗ **mlpack::det::Trainer** (arma::mat &dataset, const size_t folds, const bool useVolumeReg=false, const size_t maxLeafSize=10, const size_t minLeafSize=5, const std::string unprunedTreeOutput="")

  Train the optimal decision tree using cross-validation with the given number of folds.

22.118.1 Detailed Description

Author

Parikshit Ram (pram@cc.gatech.edu)

This file implements functions to perform different tasks with the Density Tree class.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

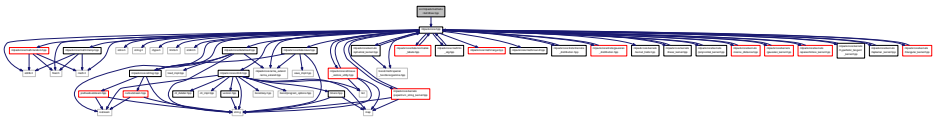
MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see http://www.gnu.org/licenses/.

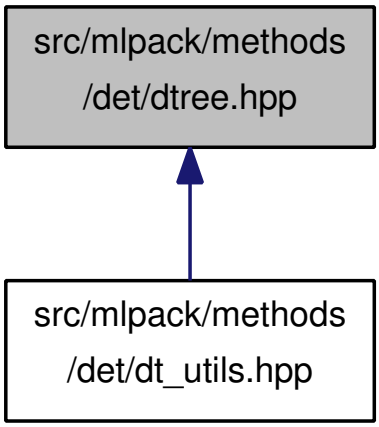
Definition in file **dt_utils.hpp**.

22.119 src/mlpack/methods/det/dtree.hpp File Reference

Include dependency graph for dtree.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class **mlpack::det::DTree**

  A density estimation tree is similar to both a decision tree and a space partitioning tree (like a kd-tree).

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::det**

  Density Estimation Trees.

22.119.1 Detailed Description

Author

Parikshit Ram (pram@cc.gatech.edu)

Density Estimation Tree class

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

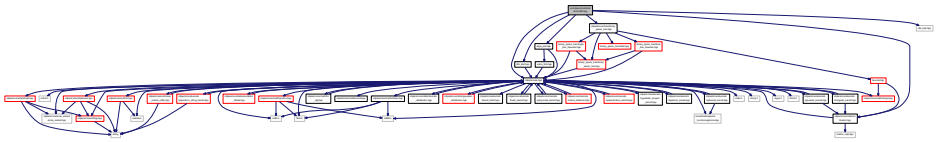
MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see http://www.gnu.org/licenses/.

Definition in file **dtree.hpp**.

22.120 src/mlpack/methods/emst/dtb.hpp File Reference

Include dependency graph for dtb.hpp:



Classes

- class **mlpack::emst::DualTreeBoruvka**< **MetricType, TreeType** >

  Performs the MST calculation using the Dual-Tree Boruvka algorithm, using any type of tree.
- struct **mlpack::emst::DualTreeBoruvka**< **MetricType, TreeType** >**::SortEdgesHelper**

  For sorting the edge list after the computation.

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.

- **mlpack::emst**

  Euclidean Minimum Spanning Trees.

22.120.1 Detailed Description

Author

Bill March (march@gatech.edu)

Contains an implementation of the DualTreeBoruvka algorithm for finding a Euclidean Minimum Spanning Tree using the kd-tree data structure.

```
@inproceedings{
  author = {March, W.B., Ram, P., and Gray, A.G.},
  title = {{Fast Euclidean Minimum Spanning Tree: Algorithm, Analysis,
     Applications.}},
  booktitle = {Proceedings of the 16th ACM SIGKDD International Conference
     on Knowledge Discovery and Data Mining}
  series = {KDD 2010},
  year = {2010}
}
```

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see http://www.gnu.org/licenses/.

Author

Bill March (march@gatech.edu)

Tree traverser rules for the DualTreeBoruvka algorithm.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see http://www.gnu.org/licenses/.

Author

Bill March (march@gatech.edu)

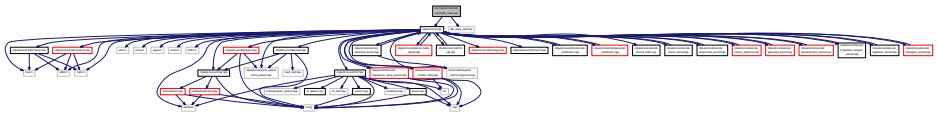
DTBStat is the StatisticType used by trees when performing EMST.

This file is part of MLPACK 1.0.8.

Definition in file **dtb.hpp**.

22.121 src/mlpack/methods/emst/dtb_rules.hpp File Reference

Include dependency graph for dtb_rules.hpp:



Classes

- class **mlpack::emst::DTBRules**< **MetricType, TreeType** >

Namespaces

- **mlpack**

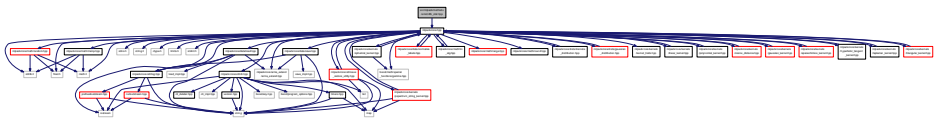
  Linear algebra utility functions, generally performed on matrices or vectors.

- **mlpack::emst**

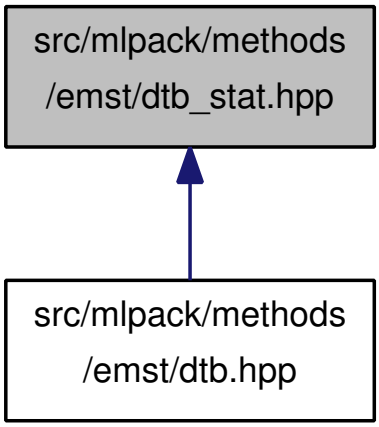
  Euclidean Minimum Spanning Trees.

22.122 src/mlpack/methods/emst/dtb_stat.hpp File Reference

Include dependency graph for dtb_stat.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class **mlpack::emst::DTBStat**

  A statistic for use with MLPACK trees, which stores the upper bound on distance to nearest neighbors and the component which this node belongs to.

Namespaces

- **mlpack**

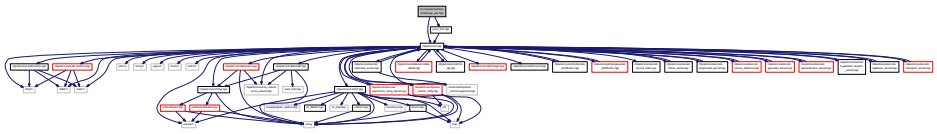
  Linear algebra utility functions, generally performed on matrices or vectors.

- **mlpack::emst**

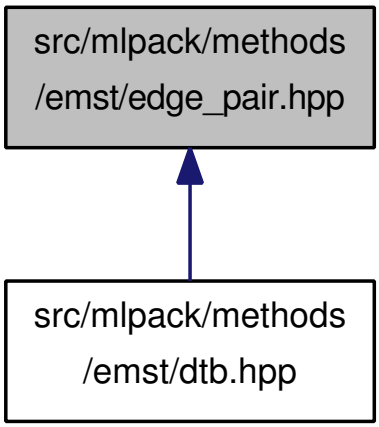
  Euclidean Minimum Spanning Trees.

22.123 src/mlpack/methods/emst/edge_pair.hpp File Reference

Include dependency graph for edge_pair.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class **mlpack::emst::EdgePair**

  An edge pair is simply two indices and a distance.

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::emst**

  Euclidean Minimum Spanning Trees.

22.123.1 Detailed Description

Author

Bill March (`march@gatech.edu`)

This file contains utilities necessary for all of the minimum spanning tree algorithms.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

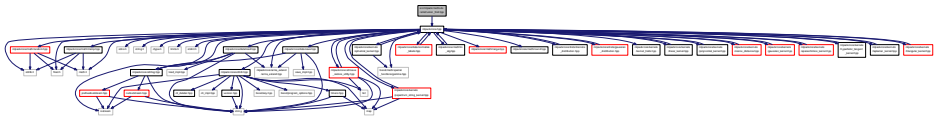
MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

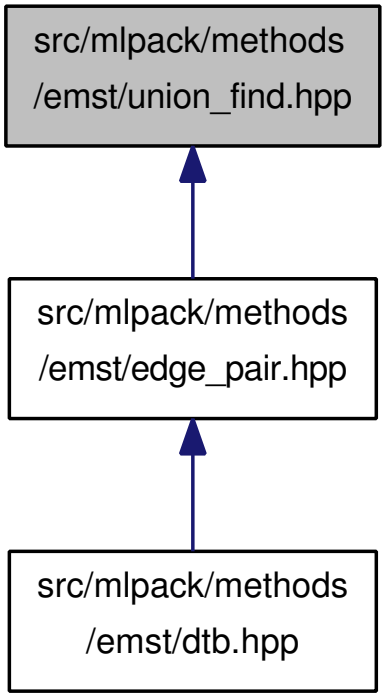
Definition in file **edge_pair.hpp**.

22.124 src/mlpack/methods/emst/union_find.hpp File Reference

Include dependency graph for union_find.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class **mlpack::emst::UnionFind**

  A Union-Find data structure.

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::emst**

  Euclidean Minimum Spanning Trees.

22.124.1 Detailed Description

Author

Bill March (march@gatech.edu)

Implements a union-find data structure. This structure tracks the components of a graph. Each point in the graph is initially in its own component. Calling unionfind.Union(x, y) unites the components indexed by x and y. unionfind.Find(x) returns the index of the component containing point x.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

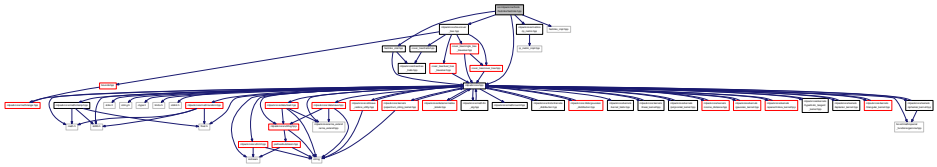
MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see http://www.gnu.org/licenses/.

Definition in file **union_find.hpp**.

22.125 src/mlpack/methods/fastmks/fastmks.hpp File Reference

Include dependency graph for fastmks.hpp:



Classes

- class **mlpack::fastmks::FastMKS**< **KernelType, TreeType** >

  An implementation of fast exact max-kernel search.

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::fastmks**

  Fast max-kernel search.

22.125.1 Detailed Description

Author

Ryan Curtin

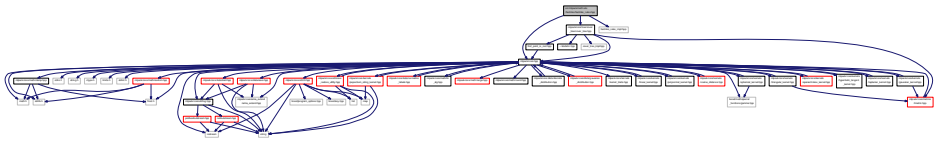
Definition of the FastMKS class, which implements fast exact max-kernel search.

This file is part of MLPACK 1.0.8.

Definition in file **fastmks.hpp**.

22.126 src/mlpack/methods/fastmks/fastmks_rules.hpp File Reference

Include dependency graph for fastmks_rules.hpp:



Classes

- class **mlpack::fastmks::FastMKSRules**< **KernelType, TreeType** >

  *The base case and pruning rules for **FastMKS** (p. 185) (fast max-kernel search).*

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::fastmks**

  Fast max-kernel search.

22.126.1 Detailed Description

Author

Ryan Curtin

Rules for the single or dual tree traversal for fast max-kernel search.

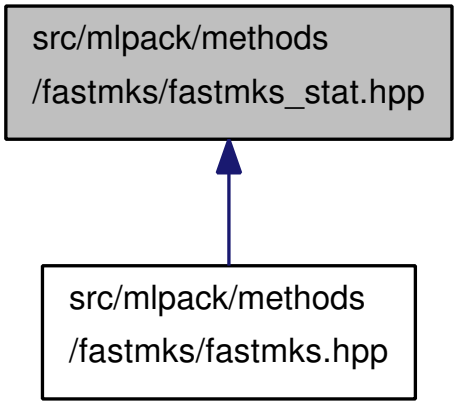
This file is part of MLPACK 1.0.8.

Definition in file **fastmks_rules.hpp**.

22.127 src/mlpack/methods/fastmks/fastmks_stat.hpp File Reference

Include dependency graph for fastmks_stat.hpp:

This graph shows which files directly or indirectly include this file:



Classes

- class **mlpack::fastmks::FastMKSStat**

  *The statistic used in trees with **FastMKS** (p. 185).*

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::fastmks**

  Fast max-kernel search.

22.127.1 Detailed Description

Author

Ryan Curtin

The statistic used in trees with FastMKS.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

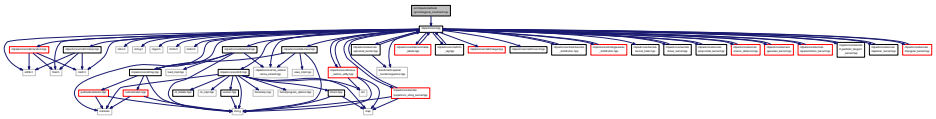
MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

Definition in file **fastmks_stat.hpp**.

22.128 src/mlpack/methods/gmm/diagonal_constraint.hpp File Reference

Include dependency graph for diagonal_constraint.hpp:



Classes

- class **mlpack::gmm::DiagonalConstraint**

  Force a covariance matrix to be diagonal.

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::gmm**

  Gaussian Mixture Models.

22.128.1 Detailed Description

Author

Ryan Curtin

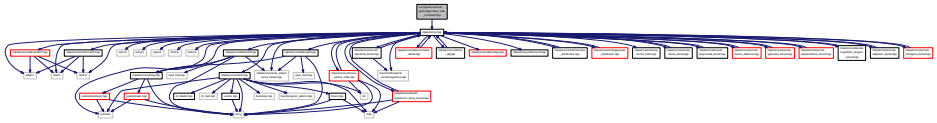
Constrain a covariance matrix to be diagonal.

This file is part of MLPACK 1.0.8.

Definition in file **diagonal_constraint.hpp**.

22.129 src/mlpack/methods/gmm/eigenvalue_ratio_constraint.hpp File Reference

Include dependency graph for eigenvalue_ratio_constraint.hpp:



Classes

- class **mlpack::gmm::EigenvalueRatioConstraint**

  Given a vector of eigenvalue ratios, ensure that the covariance matrix always has those eigenvalue ratios.

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::gmm**

  Gaussian Mixture Models.

22.129.1 Detailed Description

Author

Ryan Curtin

Constrain a covariance matrix to have a certain ratio of eigenvalues.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

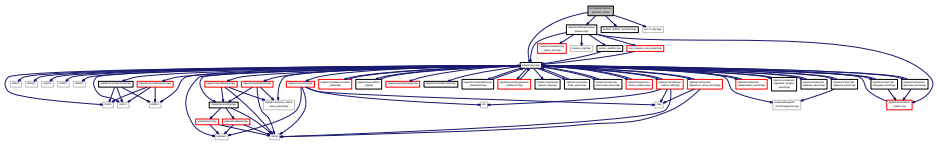
MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

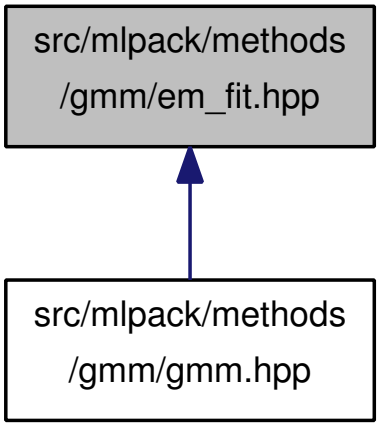
Definition in file **eigenvalue_ratio_constraint.hpp**.

22.130 src/mlpack/methods/gmm/em_fit.hpp File Reference

Include dependency graph for em_fit.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class **mlpack::gmm::EMFit**< **InitialClusteringType, CovarianceConstraintPolicy** >

  *This class contains methods which can fit a **GMM** (p. 208) to observations using the EM algorithm.*

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::gmm**

  Gaussian Mixture Models.

22.130.1 Detailed Description

Author

Ryan Curtin

Utility class to fit a GMM using the EM algorithm. Used by GMM::Estimate<>().

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

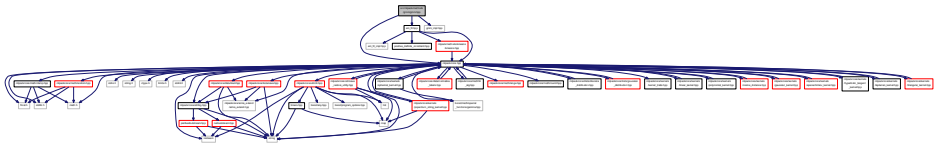
MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

Definition in file **em_fit.hpp**.

22.131 src/mlpack/methods/gmm/gmm.hpp File Reference

Include dependency graph for gmm.hpp:



Classes

- class **mlpack::gmm::GMM**< **FittingType** >

  *A Gaussian Mixture Model (**GMM** (p. 208)).*

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::gmm**

  Gaussian Mixture Models.

22.131.1 Detailed Description

Author

Parikshit Ram (pram@cc.gatech.edu)

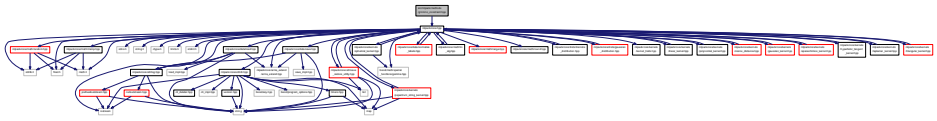
Defines a Gaussian Mixture model and estimates the parameters of the model

This file is part of MLPACK 1.0.8.

Definition in file **gmm.hpp**.

22.132 src/mlpack/methods/gmm/no_constraint.hpp File Reference

Include dependency graph for no_constraint.hpp:



Classes

- class **mlpack::gmm::NoConstraint**

  This class enforces no constraint on the covariance matrix.

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::gmm**

  Gaussian Mixture Models.

22.132.1 Detailed Description

Author

Ryan Curtin

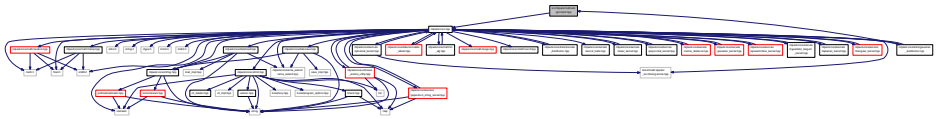
No constraint on the covariance matrix.

This file is part of MLPACK 1.0.8.

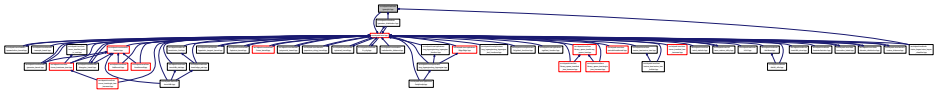
Definition in file **no_constraint.hpp**.

22.133 src/mlpack/methods/gmm/phi.hpp File Reference

Include dependency graph for phi.hpp:



This graph shows which files directly or indirectly include this file:



Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::gmm**

  Gaussian Mixture Models.

Functions

- double **mlpack::gmm::phi** (const double x, const double mean, const double var)

  Calculates the univariate Gaussian probability density function.
- double **mlpack::gmm::phi** (const arma::vec &x, const arma::vec &mean, const arma::mat &cov)

  Calculates the multivariate Gaussian probability density function.
- double **mlpack::gmm::phi** (const arma::vec &x, const arma::vec &mean, const arma::mat &cov, const std::vector< arma::mat > &d_cov, arma::vec &g_mean, arma::vec &g_cov)

  Calculates the multivariate Gaussian probability density function and also the gradients with respect to the mean and the variance.
- void **mlpack::gmm::phi** (const arma::mat &x, const arma::vec &mean, const arma::mat &cov, arma::vec &probabilities)

  Calculates the multivariate Gaussian probability density function for each data point (column) in the given matrix, with respect to the given mean and variance.

22.133.1 Detailed Description

Author

Parikshit Ram (pram@cc.gatech.edu)

This file computes the Gaussian probability density function

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

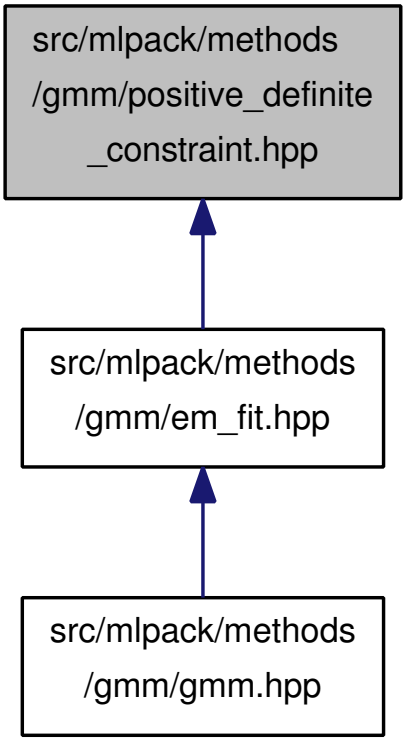
MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see http://www.gnu.org/licenses/.

Definition in file **phi.hpp**.

22.134 src/mlpack/methods/gmm/positive_definite_constraint.hpp File Reference

This graph shows which files directly or indirectly include this file:



Classes

- class **mlpack::gmm::PositiveDefiniteConstraint**

  Given a covariance matrix, force the matrix to be positive definite.

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.

- **mlpack::gmm**

  Gaussian Mixture Models.

22.134.1 Detailed Description

Author

Ryan Curtin

Restricts a covariance matrix to being positive definite.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

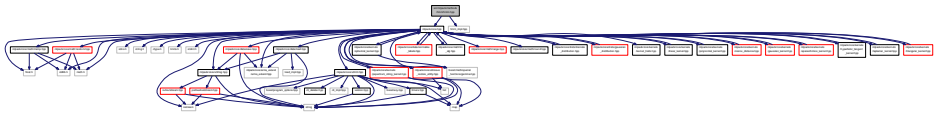
MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

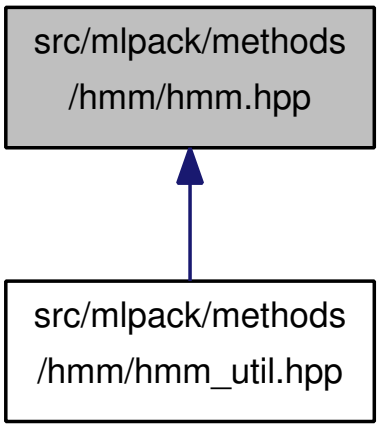
Definition in file **positive_definite_constraint.hpp**.

22.135 src/mlpack/methods/hmm/hmm.hpp File Reference

Include dependency graph for hmm.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class **mlpack::hmm::HMM**< **Distribution** >

  A class that represents a Hidden Markov Model with an arbitrary type of emission distribution.

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::hmm**

  Hidden Markov Models.

22.135.1 Detailed Description

Author

Ryan Curtin
Tran Quoc Long

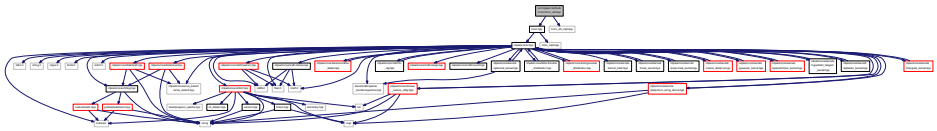
Definition of HMM class.

This file is part of MLPACK 1.0.8.

Definition in file **hmm.hpp**.

22.136 src/mlpack/methods/hmm/hmm_util.hpp File Reference

Include dependency graph for hmm_util.hpp:



Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.

- **mlpack::hmm**

  Hidden Markov Models.

Functions

- template<typename Distribution >
  void **mlpack::hmm::LoadHMM** (HMM< Distribution > &hmm, util::SaveRestoreUtility &sr)

  *Load an **HMM** (p. 221) from file.*

- template<typename Distribution >
  void **mlpack::hmm::SaveHMM** (const HMM< Distribution > &hmm, util::SaveRestoreUtility &sr)

  *Save an **HMM** (p. 221) to file.*

22.136.1 Detailed Description

Author

Ryan Curtin

Save/load utilities for HMMs. This should be eventually merged into the HMM class itself.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.-gnu.org/licenses/`.

Definition in file **hmm_util.hpp**.

22.137 src/mlpack/methods/kernel_pca/kernel_pca.hpp File Reference

Include dependency graph for kernel_pca.hpp:

Classes

- class **mlpack::kpca::KernelPCA< KernelType >**

  This class performs kernel principal components analysis (Kernel PCA), for a given kernel.

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::kpca**

22.137.1 Detailed Description

Author

Ajinkya Kale

Defines the KernelPCA class to perform Kernel Principal Components Analysis on the specified data set.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

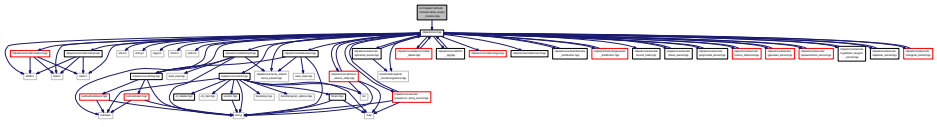
MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

Definition in file **kernel_pca.hpp**.

22.138 src/mlpack/methods/kmeans/allow_empty_clusters.hpp File Reference

Include dependency graph for allow_empty_clusters.hpp:



Classes

- class **mlpack::kmeans::AllowEmptyClusters**

  Policy which allows K-Means to create empty clusters without any error being reported.

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.

- **mlpack::kmeans**

  K-Means clustering.

22.138.1 Detailed Description

Author

Ryan Curtin

This very simple policy is used when K-Means is allowed to return empty clusters.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

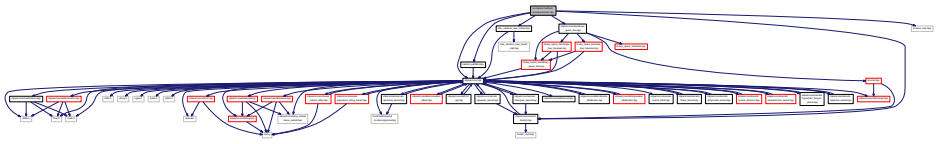
MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

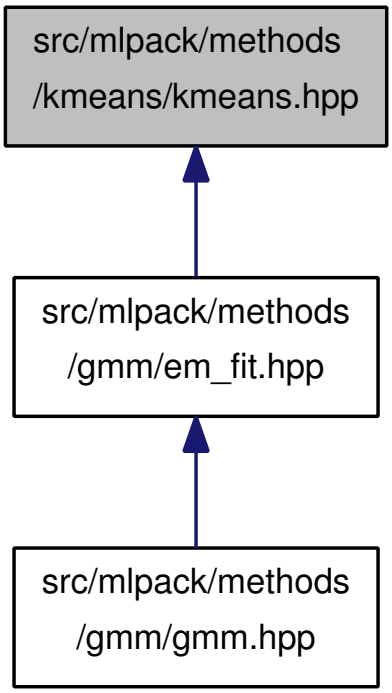
Definition in file **allow_empty_clusters.hpp**.

22.139 src/mlpack/methods/kmeans/kmeans.hpp File Reference

Include dependency graph for kmeans.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class **mlpack::kmeans::KMeans< MetricType, InitialPartitionPolicy, EmptyClusterPolicy >**

  This class implements K-Means clustering.

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::kmeans**

  K-Means clustering.

22.139.1 Detailed Description

Author

Parikshit Ram (pram@cc.gatech.edu)

K-Means clustering.

This file is part of MLPACK 1.0.8.

MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

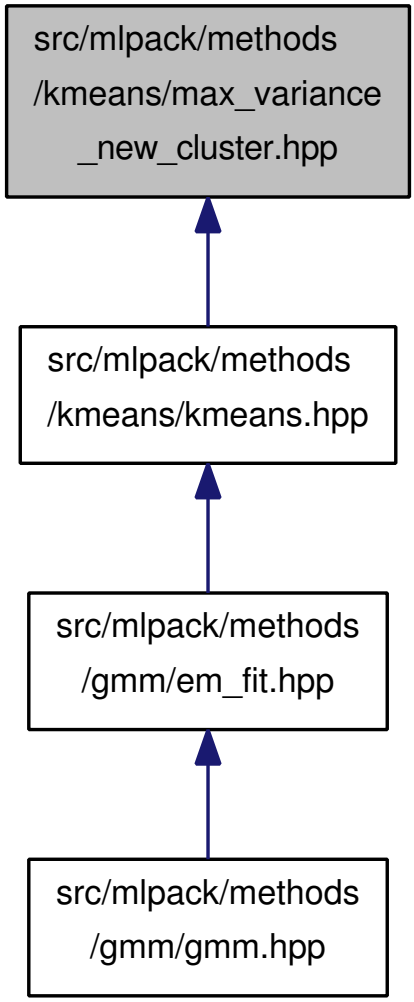
You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

Definition in file **kmeans.hpp**.

22.140 src/mlpack/methods/kmeans/max_variance_new_cluster.hpp File Reference

Include dependency graph for max_variance_new_cluster.hpp:

This graph shows which files directly or indirectly include this file:



Classes

- class **mlpack::kmeans::MaxVarianceNewCluster**

  When an empty cluster is detected, this class takes the point furthest from the centroid of the cluster with maximum variance as a new cluster.

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.

- **mlpack::kmeans**

  K-Means clustering.

22.140.1 Detailed Description

Author

Ryan Curtin

An implementation of the EmptyClusterPolicy policy class for K-Means. When an empty cluster is detected, the point furthest from the centroid of the cluster with maximum variance is taken to be a new cluster.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

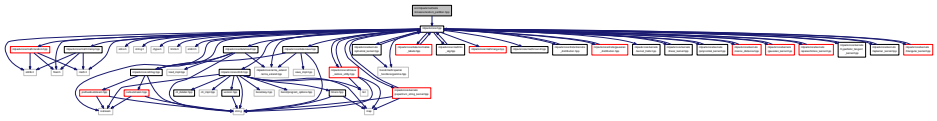
MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

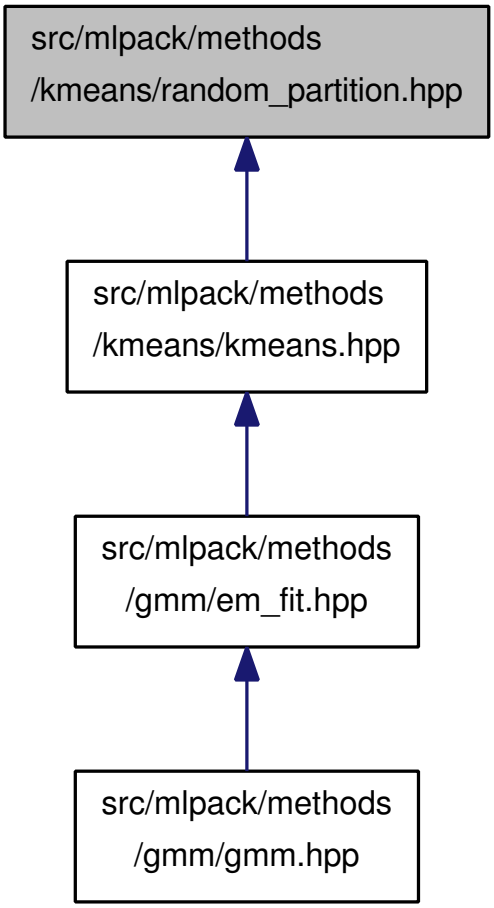
Definition in file **max_variance_new_cluster.hpp**.

22.141 src/mlpack/methods/kmeans/random_partition.hpp File Reference

Include dependency graph for random_partition.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class **mlpack::kmeans::RandomPartition**

  A very simple partitioner which partitions the data randomly into the number of desired clusters.

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::kmeans**

  K-Means clustering.

22.141.1 Detailed Description

Author

Ryan Curtin

Very simple partitioner which partitions the data randomly into the number of desired clusters. Used as the default InitialPartitionPolicy for KMeans.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.-gnu.org/licenses/`.

Definition in file **random_partition.hpp**.

22.142 src/mlpack/methods/kmeans/refined_start.hpp File Reference

Include dependency graph for refined_start.hpp:

Classes

- class **mlpack::kmeans::RefinedStart**

  A refined approach for choosing initial points for k-means clustering.

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::kmeans**

  K-Means clustering.

22.142.1 Detailed Description

Author

Ryan Curtin

An implementation of Bradley and Fayyad's "Refining Initial Points for K-Means clustering". This class is meant to provide better initial points for the k-means algorithm.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

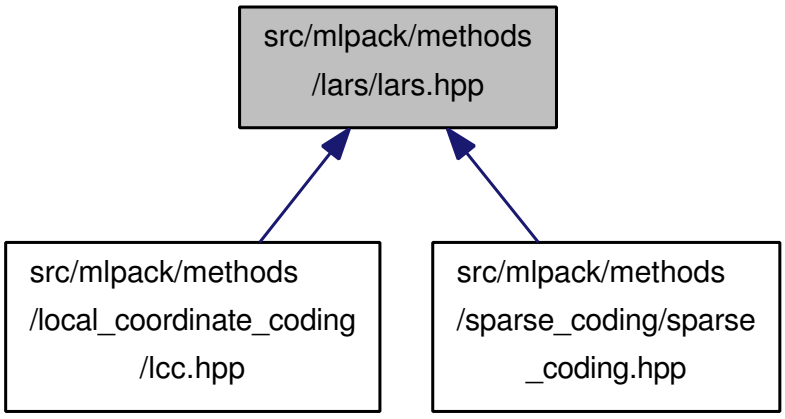
You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

Definition in file **refined_start.hpp**.

22.143 src/mlpack/methods/lars/lars.hpp File Reference

Include dependency graph for lars.hpp:

This graph shows which files directly or indirectly include this file:



Classes

- class **mlpack::regression::LARS**

  *An implementation of **LARS** (p. 429), a stage-wise homotopy-based algorithm for l1-regularized linear regression (LASSO) and l1+l2 regularized linear regression (Elastic Net).*

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::regression**

  Regression methods.

22.143.1 Detailed Description

Author

Nishant Mehta (niche)

Definition of the LARS class, which performs Least Angle Regression and the LASSO.

Only minor modifications of LARS are necessary to handle the constrained version of the problem:

$$\min_{\beta} 0.5||X\beta - y||_2^2 + 0.5\lambda_2||\beta||_2^2$$

subject to $||\beta||_1 <= \tau$

Although this option currently is not implemented, it will be implemented very soon.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

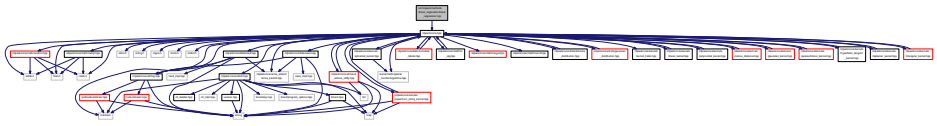
MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.-gnu.org/licenses/`.

Definition in file **lars.hpp**.

22.144 src/mlpack/methods/linear_regression/linear_regression.hpp File Reference

Include dependency graph for linear_regression.hpp:



Classes

- class **mlpack::regression::LinearRegression**

  A simple linear regression algorithm using ordinary least squares.

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::regression**

  Regression methods.

22.144.1 Detailed Description

Author

James Cline

Simple least-squares linear regression.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

Definition in file **linear_regression.hpp**.

22.145 src/mlpack/methods/local_coordinate_coding/lcc.hpp File Reference

Include dependency graph for lcc.hpp:

Classes

- class **mlpack::lcc::LocalCoordinateCoding**< **DictionaryInitializer** >

  An implementation of Local Coordinate Coding (LCC) that codes data which approximately lives on a manifold using a variation of l1-norm regularized sparse coding; in LCC, the penalty on the absolute value of each point's coefficient for each atom is weighted by the squared distance of that point to that atom.

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::lcc**

22.145.1 Detailed Description

Author

Nishant Mehta

Definition of the LocalCoordinateCoding class, which performs the Local Coordinate Coding algorithm.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

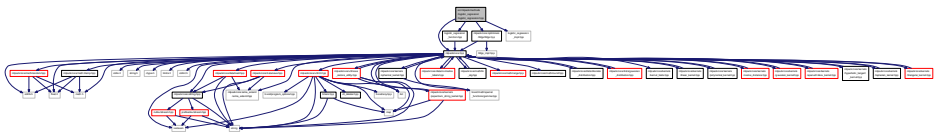
MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

Definition in file **lcc.hpp**.

22.146 src/mlpack/methods/logistic_regression/logistic_regression.hpp File Reference

Include dependency graph for logistic_regression.hpp:



Classes

- class **mlpack::regression::LogisticRegression**< **OptimizerType** >

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::regression**

  Regression methods.

22.146.1 Detailed Description

Author

Sumedh Ghaisas

The LogisticRegression class, which implements logistic regression. This implements supports L2-regularization.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

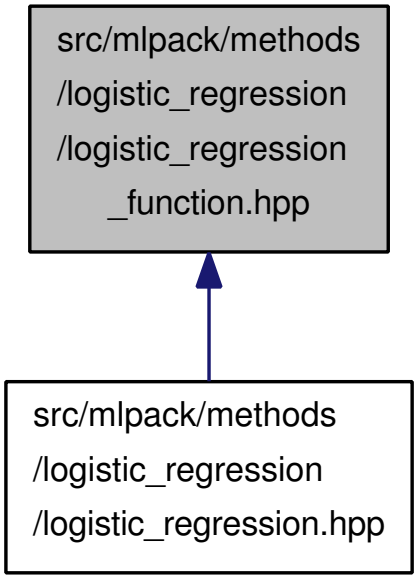
You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

Definition in file **logistic_regression.hpp**.

22.147 src/mlpack/methods/logistic_regression/logistic_regression_function.hpp File Reference

Include dependency graph for logistic_regression_function.hpp:

This graph shows which files directly or indirectly include this file:



Classes

- class **mlpack::regression::LogisticRegressionFunction**

  The log-likelihood function for the logistic regression objective function.

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::regression**

  Regression methods.

22.147.1 Detailed Description

Author

Sumedh Ghaisas

Implementation of the logistic regression function, which is meant to be optimized by a separate optimizer class that takes LogisticRegressionFunction as its FunctionType class.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

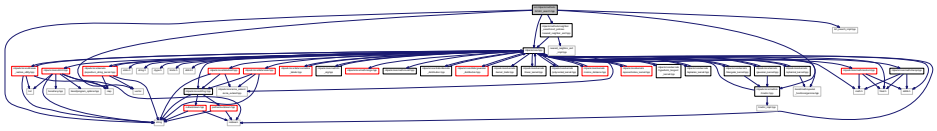
MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

Definition in file **logistic_regression_function.hpp**.

22.148 src/mlpack/methods/lsh/lsh_search.hpp File Reference

Include dependency graph for lsh_search.hpp:



Classes

- class **mlpack::neighbor::LSHSearch**< **SortPolicy** >

  *The **LSHSearch** (p. 307) class – This class builds a hash on the reference set and uses this hash to compute the distance-approximate nearest-neighbors of the given queries.*

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.

- **mlpack::neighbor**

  Neighbor-search routines.

22.148.1 Detailed Description

Author

Parikshit Ram

Defines the LSHSearch class, which performs an approximate nearest neighbor search for a queries in a query set over a given dataset using Locality-sensitive hashing with 2-stable distributions.

The details of this method can be found in the following paper:

{datar2004locality, title={Locality-sensitive hashing scheme based on p-stable distributions}, author={Datar, M. and Immorlica, N. and Indyk, P. and Mirrokni, V.S.}, booktitle= {Proceedings of the 12th Annual Symposium on Computational Geometry}, pages={253–262}, year={2004}, organization={ACM} }

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

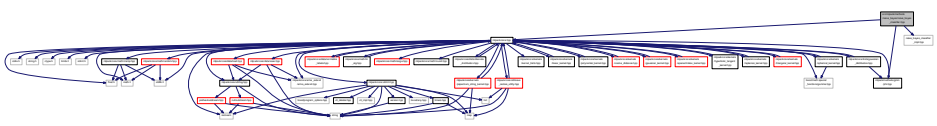
MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

Definition in file **lsh_search.hpp**.

22.149 src/mlpack/methods/naive_bayes/naive_bayes_classifier.hpp File Reference

Include dependency graph for naive_bayes_classifier.hpp:



Classes

- class **mlpack::naive_bayes::NaiveBayesClassifier**< **MatType** >

  The simple Naive Bayes classifier.

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::naive_bayes**

  The Naive Bayes Classifier.

22.149.1 Detailed Description

Author

Parikshit Ram (pram@cc.gatech.edu)

A Naive Bayes Classifier which parametrically estimates the distribution of the features. It is assumed that the features have been sampled from a Gaussian PDF.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

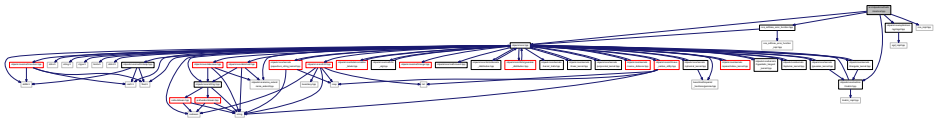
MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see http://www.gnu.org/licenses/.

Definition in file **naive_bayes_classifier.hpp**.

22.150 src/mlpack/methods/nca/nca.hpp File Reference

Include dependency graph for nca.hpp:



Classes

- class **mlpack::nca::NCA**< **MetricType, OptimizerType** >

  An implementation of Neighborhood Components Analysis, both a linear dimensionality reduction technique and a distance learning technique.

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::nca**

  Neighborhood Components Analysis.

22.150.1 Detailed Description

Author

Ryan Curtin

Declaration of NCA class (Neighborhood Components Analysis).

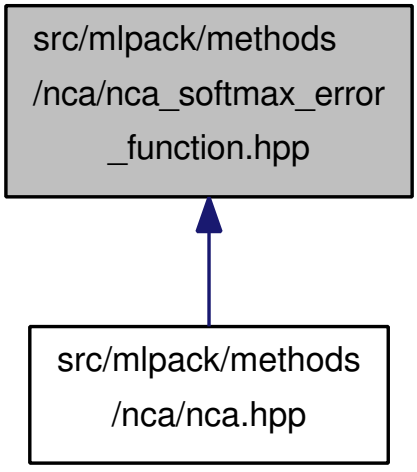
This file is part of MLPACK 1.0.8.

Definition in file **nca.hpp**.

22.151 src/mlpack/methods/nca/nca_softmax_error_function.hpp File Reference

Include dependency graph for nca_softmax_error_function.hpp:

This graph shows which files directly or indirectly include this file:



Classes

- class **mlpack::nca::SoftmaxErrorFunction**< **MetricType** >

The "softmax" stochastic neighbor assignment probability function.

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.

- **mlpack::nca**

  Neighborhood Components Analysis.

22.151.1 Detailed Description

Author

Ryan Curtin

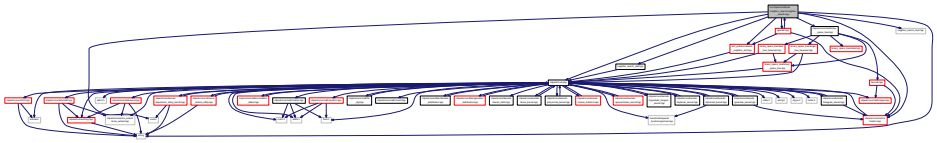
Implementation of the stochastic neighbor assignment probability error function (the "softmax error").

This file is part of MLPACK 1.0.8.

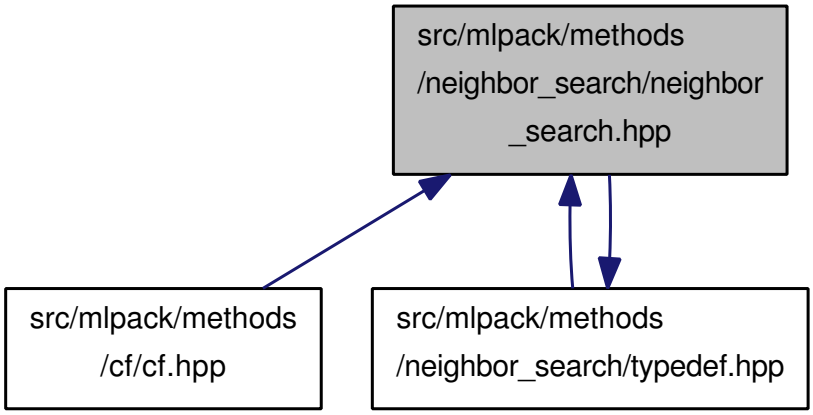
Definition in file **nca_softmax_error_function.hpp**.

22.152 src/mlpack/methods/neighbor_search/neighbor_search.hpp File Reference

Include dependency graph for neighbor_search.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class **mlpack::neighbor::NeighborSearch< SortPolicy, MetricType, TreeType >**

  *The **NeighborSearch** (p. 317) class is a template class for performing distance-based neighbor searches.*

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::neighbor**

  Neighbor-search routines.

22.152.1 Detailed Description

Author

Ryan Curtin

Defines the NeighborSearch class, which performs an abstract nearest-neighbor-like query on two datasets.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

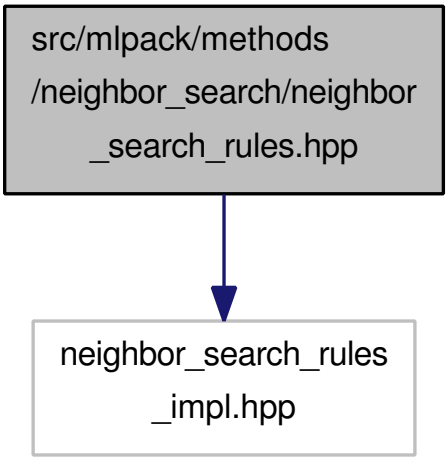
MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

Definition in file **neighbor_search.hpp**.

22.153 src/mlpack/methods/neighbor_search/neighbor_search_rules.hpp File Reference

Include dependency graph for neighbor_search_rules.hpp:



Classes

- class **mlpack::neighbor::NeighborSearchRules**< **SortPolicy, MetricType, TreeType** >

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::neighbor**

  Neighbor-search routines.

22.153.1 Detailed Description

Author

Ryan Curtin

Defines the pruning rules and base case rules necessary to perform a tree-based search (with an arbitrary tree) for the NeighborSearch class.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

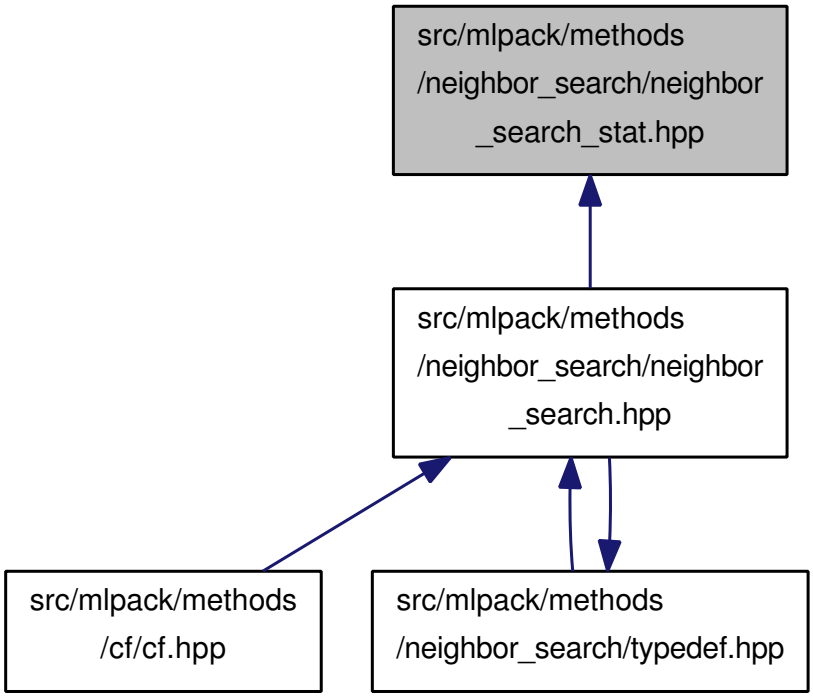
You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

Definition in file **neighbor_search_rules.hpp**.

22.154 src/mlpack/methods/neighbor_search/neighbor_search_stat.hpp File Reference

Include dependency graph for neighbor_search_stat.hpp:

This graph shows which files directly or indirectly include this file:



Classes

- class **mlpack::neighbor::NeighborSearchStat**< **SortPolicy** >

  Extra data for each node in the tree.

Namespaces

- **mlpack**

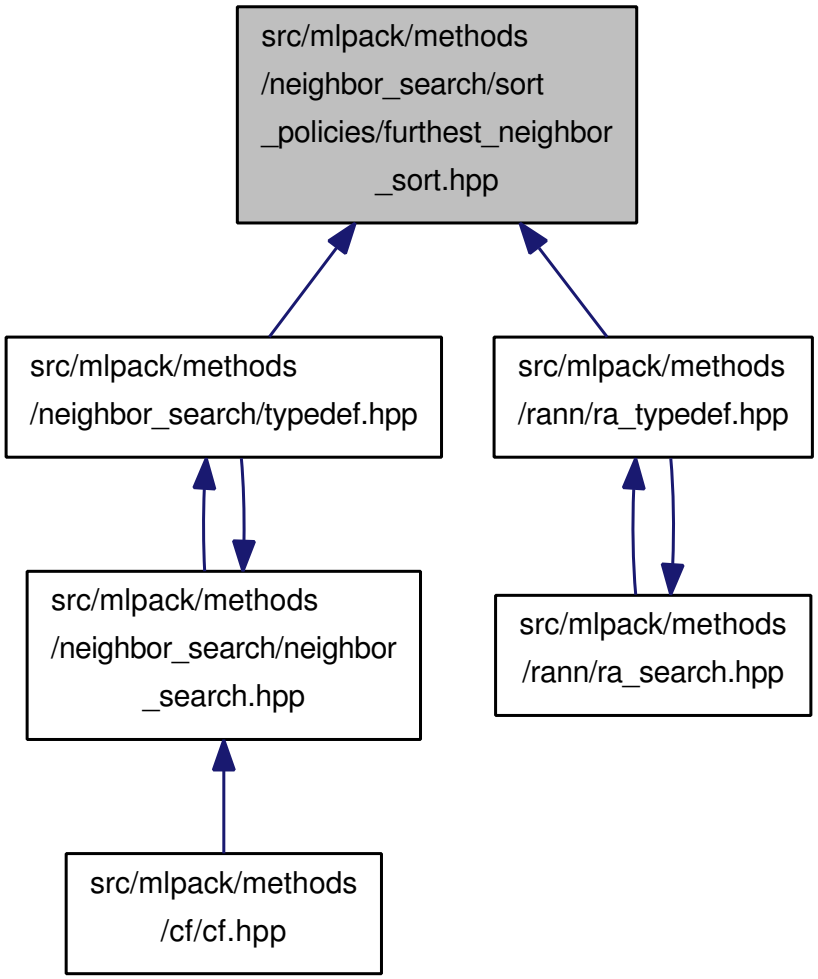
  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::neighbor**

  Neighbor-search routines.

22.155 src/mlpack/methods/neighbor_search/sort_policies/furthest_neighbor_sort.hpp File Reference

Include dependency graph for furthest_neighbor_sort.hpp:

This graph shows which files directly or indirectly include this file:



Classes

- class **mlpack::neighbor::FurthestNeighborSort**

  *This class implements the necessary methods for the SortPolicy template parameter of the **NeighborSearch** (p. 317) class.*

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.

- **mlpack::neighbor**

  Neighbor-search routines.

22.155.1 Detailed Description

Author

Ryan Curtin

Implementation of the SortPolicy class for NeighborSearch; in this case, the furthest neighbors are those that are most important.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

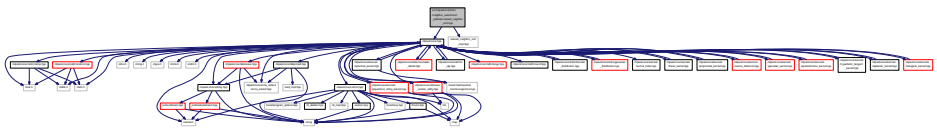
MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

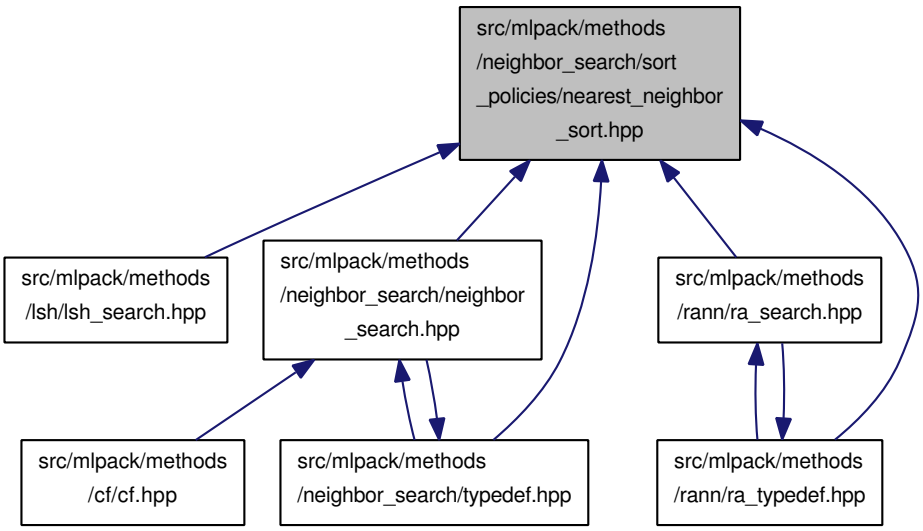
Definition in file **furthest_neighbor_sort.hpp**.

22.156 src/mlpack/methods/neighbor_search/sort_policies/nearest_neighbor_sort.hpp File Reference

Include dependency graph for nearest_neighbor_sort.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class **mlpack::neighbor::NearestNeighborSort**

  *This class implements the necessary methods for the SortPolicy template parameter of the **NeighborSearch** (p. 317) class.*

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::neighbor**

  Neighbor-search routines.

22.156.1 Detailed Description

Author

Ryan Curtin

Implementation of the SortPolicy class for NeighborSearch; in this case, the nearest neighbors are those that are most important.

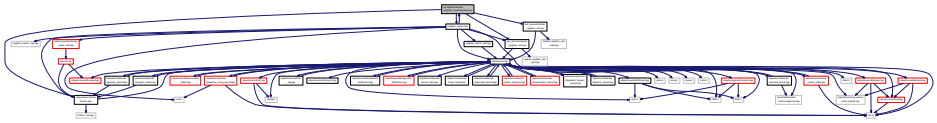
This file is part of MLPACK 1.0.8.

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

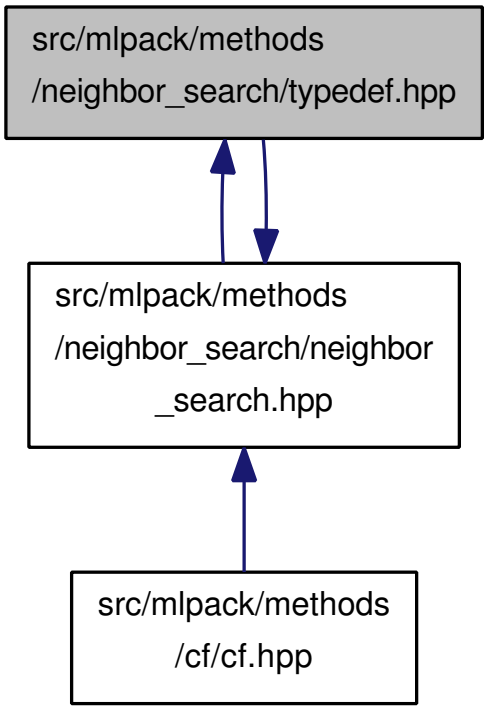
Definition in file **nearest_neighbor_sort.hpp**.

22.157 src/mlpack/methods/neighbor_search/typedef.hpp File Reference

Include dependency graph for typedef.hpp:



This graph shows which files directly or indirectly include this file:



Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.

- **mlpack::neighbor**

  Neighbor-search routines.

Typedefs

- typedef NeighborSearch < FurthestNeighborSort, metric::EuclideanDistance > **mlpack::neighbor::AllkFN**

  The AllkFN class is the all-k-furthest-neighbors method.

- typedef NeighborSearch < NearestNeighborSort, metric::EuclideanDistance > **mlpack::neighbor::AllkNN**

  The AllkNN class is the all-k-nearest-neighbors method.

22.157.1 Detailed Description

Author

Ryan Curtin

Simple typedefs describing template instantiations of the NeighborSearch class which are commonly used. This is meant to be included by neighbor_search.h but is a separate file for simplicity.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

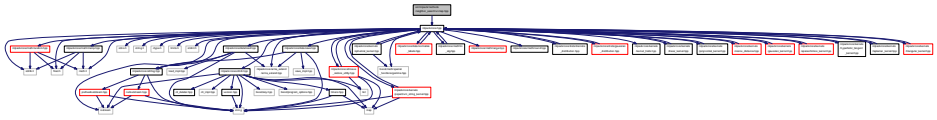
MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

Definition in file **typedef.hpp**.

22.158 src/mlpack/methods/neighbor_search/unmap.hpp File Reference

Include dependency graph for unmap.hpp:



Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.

- **mlpack::neighbor**

  Neighbor-search routines.

Functions

- void **mlpack::neighbor::Unmap** (const arma::Mat< size_t > &neighbors, const arma::mat &distances, const std::vector< size_t > &referenceMap, const std::vector< size_t > &queryMap, arma::Mat< size_t > &neighborsOut, arma::mat &distancesOut, const bool squareRoot=false)

  Assuming that the datasets have been mapped using the referenceMap and the queryMap (such as during kd-tree construction), unmap the columns of the distances and neighbors matrices into neighborsOut and distancesOut, and also unmap the entries in each row of neighbors.
- void **mlpack::neighbor::Unmap** (const arma::Mat< size_t > &neighbors, const arma::mat &distances, const std::vector< size_t > &referenceMap, arma::Mat< size_t > &neighborsOut, arma::mat &distancesOut, const bool squareRoot=false)

  Assuming that the datasets have been mapped using referenceMap (such as during kd-tree construction), unmap the columns of the distances and neighbors matrices into neighborsOut and distancesOut, and also unmap the entries in each row of neighbors.

22.158.1 Detailed Description

Author

Ryan Curtin

Convenience methods to unmap results.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

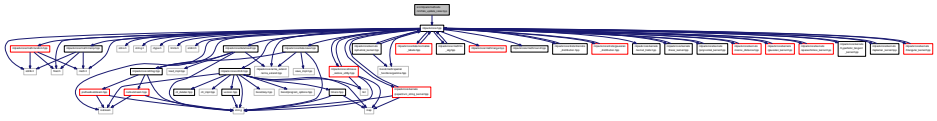
MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

Definition in file **unmap.hpp**.

22.159 src/mlpack/methods/nmf/als_update_rules.hpp File Reference

Include dependency graph for als_update_rules.hpp:



Classes

- class **mlpack::nmf::HAlternatingLeastSquaresRule**

  The update rule for the encoding matrix H.
- class **mlpack::nmf::WAlternatingLeastSquaresRule**

  The update rule for the basis matrix W.

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.

- **mlpack::nmf**

22.159.1 Detailed Description

Author

Mohan Rajendran

Update rules for the Non-negative Matrix Factorization. This follows a method titled 'Alternating Least Squares' described in the paper 'Positive Matrix Factorization: A Non-negative Factor Model with Optimal Utilization of Error Estimates of Data Values' by P. Paatero and U. Tapper. It uses least squares projection formula to reduce the error value of $\sqrt{\sum_i \sum_j (V - WH)^2}$ by alternately calculating W and H respectively while holding the other matrix constant.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

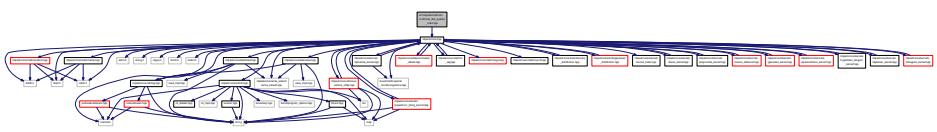
MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

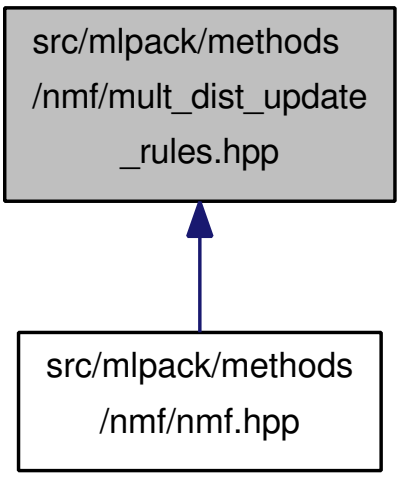
Definition in file **als_update_rules.hpp**.

22.160 src/mlpack/methods/nmf/mult_dist_update_rules.hpp File Reference

Include dependency graph for mult_dist_update_rules.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class **mlpack::nmf::HMultiplicativeDistanceRule**

  The update rule for the encoding matrix H.
- class **mlpack::nmf::WMultiplicativeDistanceRule**

  The update rule for the basis matrix W.

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::nmf**

22.160.1 Detailed Description

Author

Mohan Rajendran

Update rules for the Non-negative Matrix Factorization. This follows a method described in the paper 'Algorithms for Non-negative Matrix Factorization' by D. D. Lee and H. S. Seung. This is a multiplicative rule that ensures that the Frobenius norm $\sqrt{\sum_i \sum_j (V - WH)^2}$ is non-increasing between subsequent iterations. Both of the update rules for W and H are defined in this file.

This file is part of MLPACK 1.0.8.

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

Definition in file **mult_dist_update_rules.hpp**.

22.161 src/mlpack/methods/nmf/mult_div_update_rules.hpp File Reference

Include dependency graph for mult_div_update_rules.hpp:

Classes

- class **mlpack::nmf::HMultiplicativeDivergenceRule**

  The update rule for the encoding matrix H.
- class **mlpack::nmf::WMultiplicativeDivergenceRule**

  The update rule for the basis matrix W.

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::nmf**

22.161.1 Detailed Description

Author

Mohan Rajendran

Update rules for the Non-negative Matrix Factorization. This follows a method described in the paper 'Algorithms for Non-negative Matrix Factorization' by D. D. Lee and H. S. Seung. This is a multiplicative rule that ensures that the Kullback–Leibler divergence $\sum_i \sum_j (V_{ij} log \frac{V_{ij}}{(WH)_{ij}} - V_{ij} + (WH)_{ij})$is non-increasing between subsequent iterations. Both of the update rules for W and H are defined in this file.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

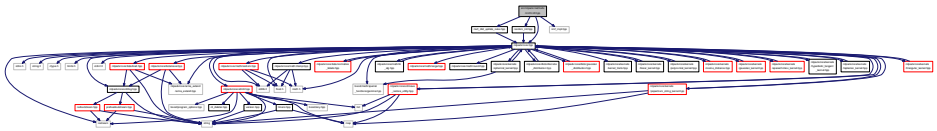
MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

Definition in file **mult_div_update_rules.hpp**.

22.162 src/mlpack/methods/nmf/nmf.hpp File Reference

Include dependency graph for nmf.hpp:



Classes

- class **mlpack::nmf::NMF**< **InitializationRule, WUpdateRule, HUpdateRule** >

  *This class implements the **NMF** (p. 353) on the given matrix V.*

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::nmf**

22.162.1 Detailed Description

Author

Mohan Rajendran

Defines the NMF class to perform Non-negative Matrix Factorization on the given matrix.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

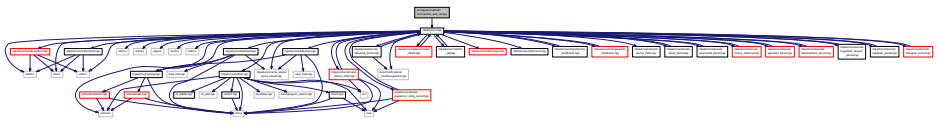
MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

Definition in file **nmf.hpp**.

22.163 src/mlpack/methods/nmf/random_acol_init.hpp File Reference

Include dependency graph for random_acol_init.hpp:



Classes

- class **mlpack::nmf::RandomAcolInitialization< p >**

  *This class initializes the W matrix of the **NMF** (p. 353) algorithm by averaging p randomly chosen columns of V.*

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.

- **mlpack::nmf**

22.163.1 Detailed Description

Author

Mohan Rajendran

Intialization rule for Non-Negative Matrix Factorization. This simple initialization is performed by the random Acol initialization introduced in the paper 'Algorithms, Initializations and Convergence' by Langville et al. This method sets each of the columns of W by averaging p randomly chosen columns of V.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

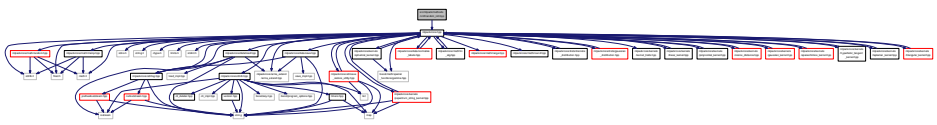
MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

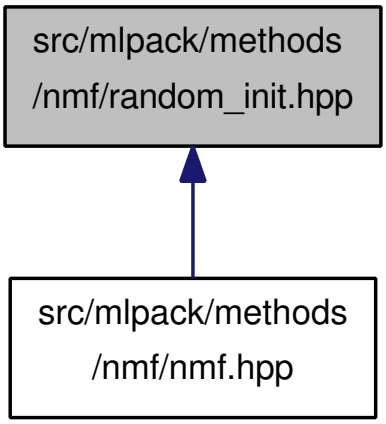
Definition in file **random_acol_init.hpp**.

22.164 src/mlpack/methods/nmf/random_init.hpp File Reference

Include dependency graph for random_init.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class **mlpack::nmf::RandomInitialization**

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::nmf**

22.164.1 Detailed Description

Author

Mohan Rajendran

Intialization rule for Non-Negative Matrix Factorization (NMF). This simple initialization is performed by assigning a random matrix to W and H.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

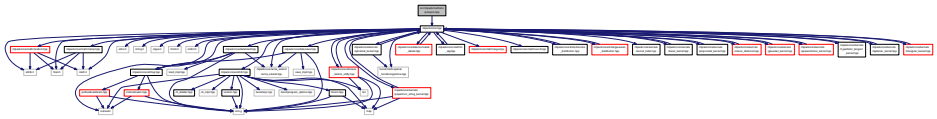
MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

Definition in file **random_init.hpp**.

22.165 src/mlpack/methods/pca/pca.hpp File Reference

Include dependency graph for pca.hpp:



Classes

- class **mlpack::pca::PCA**

  *This class implements principal components analysis (**PCA** (p. 407)).*

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::pca**

22.165.1 Detailed Description

Author

Ajinkya Kale

Defines the PCA class to perform Principal Components Analysis on the specified data set.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

Definition in file **pca.hpp**.

22.166 src/mlpack/methods/radical/radical.hpp File Reference

Include dependency graph for radical.hpp:

Classes

- class **mlpack::radical::Radical**

  An implementation of RADICAL, an algorithm for independent component analysis (ICA).

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::radical**

Functions

- void **mlpack::radical::WhitenFeatureMajorMatrix** (const arma::mat &matX, arma::mat &matXWhitened, arma::mat &matWhitening)

22.166.1 Detailed Description

Author

Nishant Mehta

Declaration of Radical class (RADICAL is Robust, Accurate, Direct ICA aLgorithm).

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

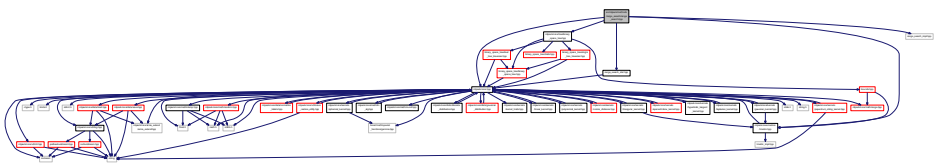
MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

Definition in file **radical.hpp**.

22.167 src/mlpack/methods/range_search/range_search.hpp File Reference

Include dependency graph for range_search.hpp:



Classes

- class **mlpack::range::RangeSearch< MetricType, TreeType >**

  *The **RangeSearch** (p. 415) class is a template class for performing range searches.*

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.

- **mlpack::range**

  Range-search routines.

22.167.1 Detailed Description

Author

Ryan Curtin

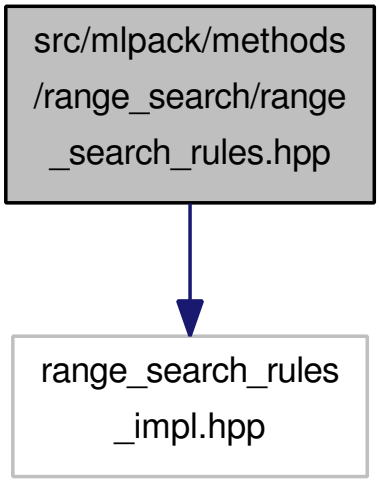
Defines the RangeSearch class, which performs a generalized range search on points.

This file is part of MLPACK 1.0.8.

Definition in file **range_search.hpp**.

22.168 src/mlpack/methods/range_search/range_search_rules.hpp File Reference

Include dependency graph for range_search_rules.hpp:



Classes

- class **mlpack::range::RangeSearchRules**< **MetricType, TreeType** >

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::range**

  Range-search routines.

22.168.1 Detailed Description

Author

Ryan Curtin

Rules for range search, so that it can be done with arbitrary tree types.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

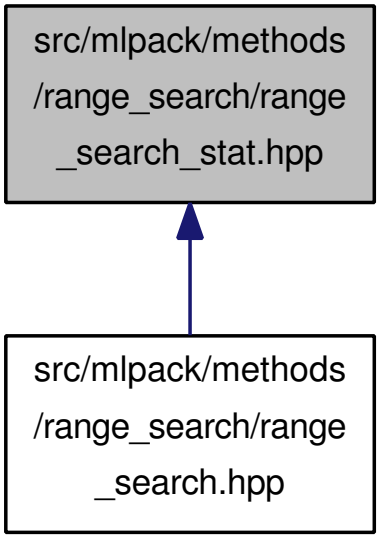
You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

Definition in file **range_search_rules.hpp**.

22.169 src/mlpack/methods/range_search/range_search_stat.hpp File Reference

Include dependency graph for range_search_stat.hpp:

This graph shows which files directly or indirectly include this file:



Classes

- class **mlpack::range::RangeSearchStat**

  *Statistic class for **RangeSearch** (p. 415), to be set to the StatisticType of the tree type that range search is being performed with.*

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::range**

  Range-search routines.

22.169.1 Detailed Description

Author

Ryan Curtin

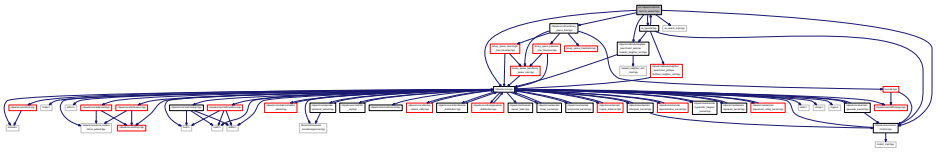
Statistic class for RangeSearch, which just holds the last visited node and the corresponding base case result.

This file is part of MLPACK 1.0.8.

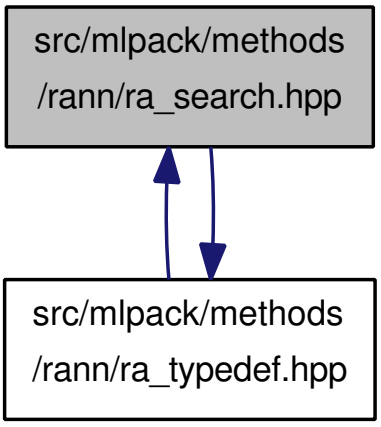
Definition in file **range_search_stat.hpp**.

22.170 src/mlpack/methods/rann/ra_search.hpp File Reference

Include dependency graph for ra_search.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class **mlpack::neighbor::RAQueryStat**< **SortPolicy** >

  Extra data for each node in the tree.
- class **mlpack::neighbor::RASearch**< **SortPolicy, MetricType, TreeType** >

*The **RASearch** (p. 334) class: This class provides a generic manner to perform rank-approximate search via random-sampling.*

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.

- **mlpack::neighbor**

  Neighbor-search routines.

22.170.1 Detailed Description

Author

Parikshit Ram

Defines the RASearch class, which performs an abstract rank-approximate nearest/farthest neighbor query on two datasets.

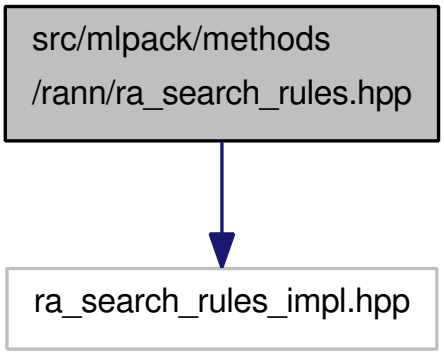
The details of this method can be found in the following paper:

{ram2009rank, title={{Rank-Approximate Nearest Neighbor Search: Retaining Meaning and Speed in High Dimensions}}, author={{Ram, P. and Lee, D. and Ouyang, H. and Gray, A. G.}}, booktitle={{Advances of Neural Information Processing Systems}}, year={2009} }

Definition in file **ra_search.hpp**.

22.171 src/mlpack/methods/rann/ra_search_rules.hpp File Reference

Include dependency graph for ra_search_rules.hpp:



Classes

- class **mlpack::neighbor::RASearchRules**< **SortPolicy, MetricType, TreeType** >

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::neighbor**

  Neighbor-search routines.

22.171.1 Detailed Description

Author

Parikshit Ram

Defines the pruning rules and base case rules necessary to perform a tree-based rank-approximate search (with an arbitrary tree) for the RASearch class.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

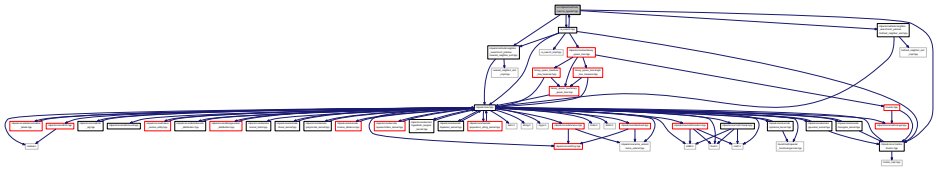
MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

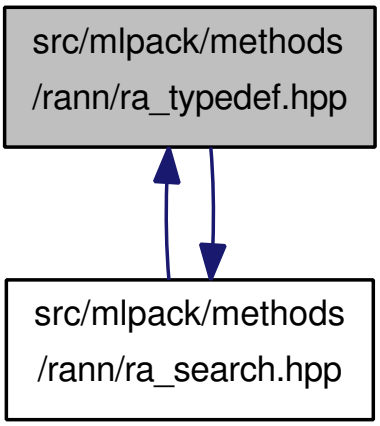
Definition in file **ra_search_rules.hpp**.

22.172 src/mlpack/methods/rann/ra_typedef.hpp File Reference

Include dependency graph for ra_typedef.hpp:



This graph shows which files directly or indirectly include this file:



Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::neighbor**

  Neighbor-search routines.

Typedefs

- typedef RASearch
  < FurthestNeighborSort > **mlpack::neighbor::AllkRAFN**

  The AllkRAFN class is the all-k-rank-approximate-farthest-neighbors method.
- typedef RASearch **mlpack::neighbor::AllkRANN**

  The AllkRANN class is the all-k-rank-approximate-nearest-neighbors method.

22.172.1 Detailed Description

Author

Parikshit Ram

Simple typedefs describing template instantiations of the RASearch class which are commonly used.

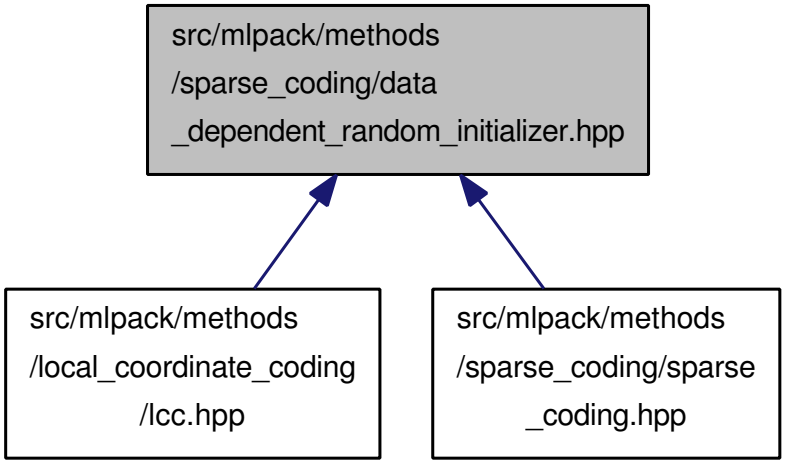
This file is part of MLPACK 1.0.8.

Definition in file **ra_typedef.hpp**.

22.173 src/mlpack/methods/sparse_coding/data_dependent_random_initializer.hpp File Reference

Include dependency graph for data_dependent_random_initializer.hpp:

This graph shows which files directly or indirectly include this file:



Classes

- class **mlpack::sparse_coding::DataDependentRandomInitializer**
  *A data-dependent random dictionary initializer for **SparseCoding** (p. 450).*

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.

- **mlpack::sparse_coding**

22.173.1 Detailed Description

Author

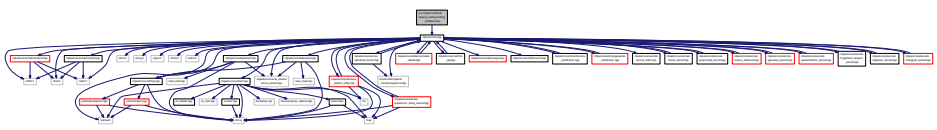
Nishant Mehta

A sensible heuristic for initializing dictionaries for sparse coding.

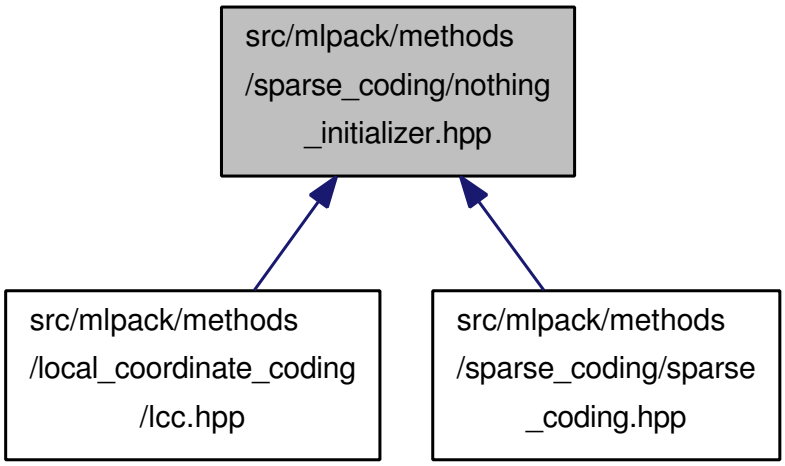
Definition in file **data_dependent_random_initializer.hpp**.

22.174 src/mlpack/methods/sparse_coding/nothing_initializer.hpp File Reference

Include dependency graph for nothing_initializer.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class **mlpack::sparse_coding::NothingInitializer**

  *A DictionaryInitializer for **SparseCoding** (p. 450) which does not initialize anything; it is useful for when the dictionary is already known and will be set with **SparseCoding::Dictionary()** (p. 453).*

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::sparse_coding**

22.174.1 Detailed Description

Author

Ryan Curtin

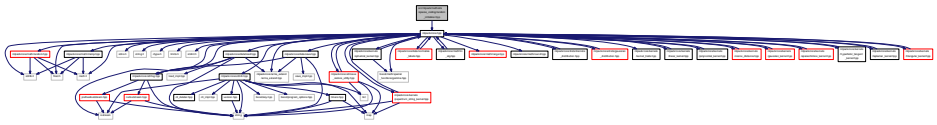
An initializer for SparseCoding which does precisely nothing. It is useful for when you have an already defined dictionary and you plan on setting it with SparseCoding::Dictionary().

This file is part of MLPACK 1.0.8.

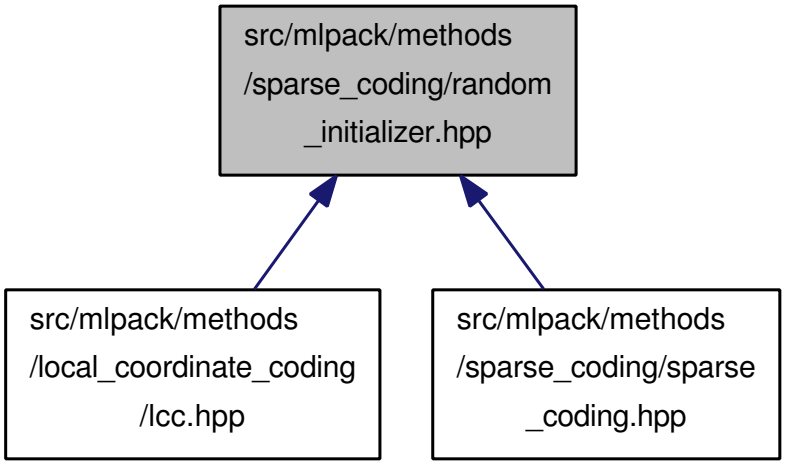
Definition in file **nothing_initializer.hpp**.

22.175 src/mlpack/methods/sparse_coding/random_initializer.hpp File Reference

Include dependency graph for random_initializer.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class **mlpack::sparse_coding::RandomInitializer**

  *A DictionaryInitializer for use with the **SparseCoding*** (p. 450) *class.*

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::sparse_coding**

22.175.1 Detailed Description

Author

Nishant Mehta

A very simple random dictionary initializer for SparseCoding; it is probably not a very good choice.

This file is part of MLPACK 1.0.8.

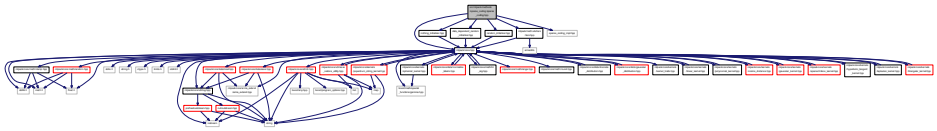
MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

Definition in file **random_initializer.hpp**.

22.176 src/mlpack/methods/sparse_coding/sparse_coding.hpp File Reference

Include dependency graph for sparse_coding.hpp:



Classes

- class **mlpack::sparse_coding::SparseCoding< DictionaryInitializer >**

  An implementation of Sparse Coding with Dictionary Learning that achieves sparsity via an l1-norm regularizer on the codes (LASSO) or an (l1+l2)-norm regularizer on the codes (the Elastic Net).

Namespaces

- **mlpack**

  Linear algebra utility functions, generally performed on matrices or vectors.
- **mlpack::sparse_coding**

22.176.1 Detailed Description

Author

Nishant Mehta

Definition of the SparseCoding class, which performs L1 (LASSO) or L1+L2 (Elastic Net)-regularized sparse coding with dictionary learning

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

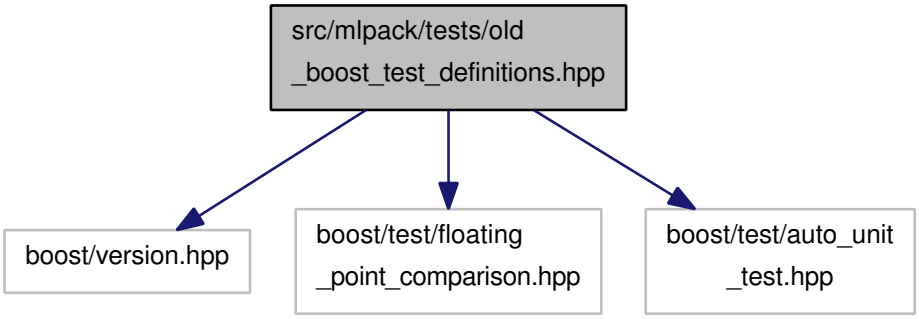
Definition in file **sparse_coding.hpp**.

22.177 src/mlpack/tests/data/data_3d_ind.txt File Reference

22.178 src/mlpack/tests/data/data_3d_mixed.txt File Reference

22.179 src/mlpack/tests/old_boost_test_definitions.hpp File Reference

Include dependency graph for old_boost_test_definitions.hpp:



Macros

- #define **BOOST_REQUIRE_GE**(L, R) BOOST_REQUIRE_EQUAL((L >= R), true)
- #define **BOOST_REQUIRE_GT**(L, R) BOOST_REQUIRE_EQUAL((L > R), true)
- #define **BOOST_REQUIRE_LE**(L, R) BOOST_REQUIRE_EQUAL((L <= R), true)
- #define **BOOST_REQUIRE_LT**(L, R) BOOST_REQUIRE_EQUAL((L < R), true)
- #define **BOOST_REQUIRE_NE**(L, R) BOOST_REQUIRE_EQUAL((L != R), true)

22.179.1 Detailed Description

Author

Ryan Curtin

Ancient Boost.Test versions don't act how we expect. This file includes the things we need to fix that.

This file is part of MLPACK 1.0.8.

MLPACK is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

MLPACK is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details (LICENSE.txt).

You should have received a copy of the GNU General Public License along with MLPACK. If not, see `http://www.gnu.org/licenses/`.

Definition in file **old_boost_test_definitions.hpp**.

22.179.2 Macro Definition Documentation

22.179.2.1 #define BOOST_REQUIRE_GE(L, R) BOOST_REQUIRE_EQUAL((L >= R), true)

Definition at line 36 of file old_boost_test_definitions.hpp.

22.179.2.2 #define BOOST_REQUIRE_GT(L, R) BOOST_REQUIRE_EQUAL((L > R), true)

Definition at line 48 of file old_boost_test_definitions.hpp.

22.179.2.3 #define BOOST_REQUIRE_LE(L, R) BOOST_REQUIRE_EQUAL((L <= R), true)

Definition at line 42 of file old_boost_test_definitions.hpp.

22.179.2.4 #define BOOST_REQUIRE_LT(L, R) BOOST_REQUIRE_EQUAL((L < R), true)

Definition at line 45 of file old_boost_test_definitions.hpp.

22.179.2.5 #define BOOST_REQUIRE_NE(L, R) BOOST_REQUIRE_EQUAL((L != R), true)

Definition at line 39 of file old_boost_test_definitions.hpp.

Index