Introduction à la réconnaissance des formes

Generated by Doxygen 1.9.1

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 3

Class Documentation

3.1 ClassMetrics Struct Reference

Metrics for evaluating classification performance for a single class.

`#include <confusion_matrix.h>`

Public Attributes

- double precision
- double recall
- double f1Score
- double accuracy

3.1.1 Detailed Description

Metrics for evaluating classification performance for a single class.

3.1.2 Member Data Documentation

3.1.2.1 accuracy

`double ClassMetrics::accuracy`

Accuracy of the class.

3.1.2.2 f1Score

`double ClassMetrics::f1Score`

F1 Score of the class.

3.1.2.3 precision

```
double ClassMetrics::precision
```

Precision of the class.

3.1.2.4 recall

```
double ClassMetrics::recall
```

Recall of the class.

The documentation for this struct was generated from the following file:

- confusion_matrix.h

3.2 Cluster Struct Reference

```
#include <kmeans.h>
```

Collaboration diagram for Cluster:

Public Attributes

- ShapeData * centroid
- int size
- ShapeData * points
- int clusterClass

3.2.1 Detailed Description

Structure to represent a cluster.

3.2.2 Member Data Documentation

3.2.2.1 centroid

```
ShapeData* Cluster::centroid
```

The centroid of the cluster.

3.2.2.2 clusterClass

```
int Cluster::clusterClass
```

Added field to store the class of the cluster.

3.2.2.3 points

```
ShapeData* Cluster::points
```

Array of points belonging to this cluster.

3.2.2.4 size

```
int Cluster::size
```

The number of elements in the cluster.

The documentation for this struct was generated from the following file:

- kmeans.h

3.3 CommandLineOptions Struct Reference

Stores command line options.

Public Attributes

- char * directory
- char * extension
- float trainingFraction
- char * method
- int p
- int k
- char * preprocessing

3.3.1 Detailed Description

Stores command line options.

3.3.2 Member Data Documentation

3.3.2.1 directory

```
char* CommandLineOptions::directory
```

Path to the directory containing data files.

3.3.2.2 extension

```
char* CommandLineOptions::extension
```

extension File extension of data files.

3.3.2.3 k

```
int CommandLineOptions::k
```

Number of neighbors/clusters.

3.3.2.4 method

```
char* CommandLineOptions::method
```

Machine learning method to use ('knn' or 'kmeans').

3.3.2.5 p

```
int CommandLineOptions::p
```

Distance metric parameter (used in k-NN and k-Means).

3.3.2.6 preprocessing

```
char* CommandLineOptions::preprocessing
```

Preprocessing method ('normalize' or 'standardize').

3.3.2.7 trainingFraction

```
float CommandLineOptions::trainingFraction
```

Fraction of data to be used for training.

The documentation for this struct was generated from the following file:

- main.c

3.4 ConfusionMatrix Struct Reference

Represents a confusion matrix for classification results.

`#include <confusion_matrix.h>`

Public Attributes

- int ** matrix
- int classCount

3.4.1 Detailed Description

Represents a confusion matrix for classification results.

3.4.2 Member Data Documentation

3.4.2.1 classCount

```
int ConfusionMatrix::classCount
```

Number of classes.

3.4.2.2 matrix

```
int** ConfusionMatrix::matrix
```

2D array representing the confusion matrix.

The documentation for this struct was generated from the following file:

- confusion_matrix.h

3.5 ConfusionMatrixMetrics Struct Reference

Container for metrics calculated from a confusion matrix.

`#include <confusion_matrix.h>`

Collaboration diagram for ConfusionMatrixMetrics:

Public Attributes

- ClassMetrics * classMetrics
- ClassMetrics overallMetrics
- int classCount

3.5.1 Detailed Description

Container for metrics calculated from a confusion matrix.

3.5.2 Member Data Documentation

3.5.2.1 classCount

```
int ConfusionMatrixMetrics::classCount
```

Number of classes.

3.5.2.2 classMetrics

```
ClassMetrics* ConfusionMatrixMetrics::classMetrics
```

Array of metrics for each class.

3.5.2.3 overallMetrics

```
ClassMetrics ConfusionMatrixMetrics::overallMetrics
```

Overall metrics computed from the entire confusion matrix.

The documentation for this struct was generated from the following file:

- confusion_matrix.h

3.6 CrossValidationMetrics Struct Reference

Structure to hold aggregated metrics from k-fold cross-validation.

```
#include <cross_validation.h>
```

Public Attributes

- double totalAccuracy
- double totalPrecision
- double totalRecall
- double totalF1Score
- int foldsProcessed

3.6.1 Detailed Description

Structure to hold aggregated metrics from k-fold cross-validation.

3.6.2 Member Data Documentation

3.6.2.1 foldsProcessed

```
int CrossValidationMetrics::foldsProcessed
```

Number of folds processed.

3.6.2.2 totalAccuracy

```
double CrossValidationMetrics::totalAccuracy
```

Total accuracy across all folds.

3.6.2.3 totalF1Score

```
double CrossValidationMetrics::totalF1Score
```

Total F1 score across all folds.

3.6.2.4 totalPrecision

```
double CrossValidationMetrics::totalPrecision
```

Total precision across all folds.

3.6.2.5 totalRecall

```
double CrossValidationMetrics::totalRecall
```

Total recall across all folds.

The documentation for this struct was generated from the following file:

- cross_validation.h

3.7 DistanceLabel Struct Reference

`#include <knn.h>`

Public Attributes

- double distance
- int label

3.7.1 Detailed Description

Structure to associate a distance with a class label. Used in the k-NN algorithm for mapping distances to training sample labels.

3.7.2 Member Data Documentation

3.7.2.1 distance

`double DistanceLabel::distance`

Distance between test and training samples.

3.7.2.2 label

`int DistanceLabel::label`

Class label of the training sample.

The documentation for this struct was generated from the following file:

- knn.h

3.8 ShapeData Struct Reference

Structure representing the data of a shape.

`#include <data_reader.h>`

Public Attributes

- int class
- int sample
- double * features
- int featureCount

3.8.1 Detailed Description

Structure representing the data of a shape.

Structure to hold split data sets: training and test sets.

3.8.2 Member Data Documentation

3.8.2.1 class

```
int ShapeData::class
```

Class identifier of the shape.

3.8.2.2 featureCount

```
int ShapeData::featureCount
```

Number of elements in the features array.

3.8.2.3 features

```
double* ShapeData::features
```

Array of feature values.

3.8.2.4 sample

```
int ShapeData::sample
```

Sample number for the shape.

The documentation for this struct was generated from the following file:

- data_reader.h

3.9 SplitData Struct Reference

Collaboration diagram for SplitData:

Public Attributes

- ShapeData * trainingSet
- int trainingSize
- ShapeData * testSet
- int testSize

3.9.1 Member Data Documentation

3.9.1.1 testSet

```
ShapeData* SplitData::testSet
```

Array of ShapeData for the test set.

3.9.1.2 testSize

```
int SplitData::testSize
```

Number of elements in the test set.

3.9.1.3 trainingSet

```
ShapeData* SplitData::trainingSet
```

Array of ShapeData for the training set.

3.9.1.4 trainingSize

```
int SplitData::trainingSize
```

Number of elements in the training set.

The documentation for this struct was generated from the following file:

- data_split.h

3.10 ThreadArgs Struct Reference

Collaboration diagram for ThreadArgs:

Public Attributes

- ShapeData * data
- int startIdx
- int endIdx
- double * min
- double * max
- int featureCount

3.10.1 Member Data Documentation

3.10.1.1 data

```
ShapeData* ThreadArgs::data
```

Pointer to the array of ShapeData.

3.10.1.2 endIdx

```
int ThreadArgs::endIdx
```

Ending index of the data segment.

3.10.1.3 featureCount

```
int ThreadArgs::featureCount
```

Number of features in each ShapeData item.

3.10.1.4 max

```
double* ThreadArgs::max
```

Array to store maximum values found by this thread.

3.10.1.5 min

```
double* ThreadArgs::min
```

Array to store minimum values found by this thread.

3.10.1.6 startIdx

```
int ThreadArgs::startIdx
```

Starting index of the data segment for this thread.

The documentation for this struct was generated from the following file:

- normalization.h

Chapter 4

File Documentation

4.1 confusion_matrix.h File Reference

Header file for confusion matrix operations.

```
#include "data_reader.h"
#include <time.h>
```

Include dependency graph for confusion_matrix.h:

4.2 cross_validation.h File Reference

Header file for implementing k-fold cross-validation with model evaluation metrics.

```
#include "data_split.h"
#include "confusion_matrix.h"
```

Include dependency graph for cross_validation.h:

Classes

- struct CrossValidationMetrics

  Structure to hold aggregated metrics from k-fold cross-validation.

Typedefs

- typedef ConfusionMatrix(∗ ModelFunction) (SplitData split)

  Typedef for a function pointer representing a model's evaluation function.

Functions

- void crossValidation (ShapeData ∗data, int dataSize, int kFolds, ModelFunction modelFunc)

  Perform k-fold cross-validation on a dataset using a specified model.
- void knnModelFunction (SplitData split)

  Model function for k-Nearest Neighbors (knn) algorithm.

4.2.1 Detailed Description

Header file for implementing k-fold cross-validation with model evaluation metrics.

4.2.2 Typedef Documentation

4.2.2.1 ModelFunction

```
ModelFunction
```

Typedef for a function pointer representing a model's evaluation function.

Parameters

split	The data split containing training and test sets.

Returns

ConfusionMatrix representing the model's performance.

4.2.3 Function Documentation

4.2.3.1 crossValidation()

```
void crossValidation (
            ShapeData * data,
            int dataSize,
            int kFolds,
            ModelFunction modelFunc )
```

Perform k-fold cross-validation on a dataset using a specified model.

This function divides the dataset into k folds, trains the model on k-1 folds, and evaluates its performance on the remaining fold. This process is repeated k times to ensure each fold serves as the test set exactly once.

Parameters

data	Pointer to the dataset.
dataSize	Number of elements in the dataset.
kFolds	Number of folds for cross-validation.
modelFunc	Function pointer to the model's evaluation function.

4.2.3.2 knnModelFunction()

```
void knnModelFunction (
            SplitData split )
```

Model function for k-Nearest Neighbors (knn) algorithm.

Parameters

split	The data split containing training and test sets.

4.3 data_reader.h File Reference

Header file for reading and processing shape data from files.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <dirent.h>
```

Include dependency graph for data_reader.h: This graph shows which files directly or indirectly include this file:

Classes

- struct ShapeData

  Structure representing the data of a shape.

Macros

- #define **SUCCESS** 0
- #define **ERR_DIR_OPEN_FAILED** 1
- #define **ERR_FILE_OPEN_FAILED** 2
- #define **ERR_MEMORY_ALLOCATION_FAILED** 3
- #define **ERR_INVALID_FILENAME** -1
- #define **ERR_UNKNOWN_FILE_TYPE** 5
- #define **ERR_FEATURES_VALUES** 6

Functions

- ShapeData ∗ readAllFiles (const char ∗directory, const char ∗extension, int ∗count)

  Reads all files with a specified extension in a given directory.
- ShapeData readFile (const char ∗filename)

  Reads and processes a single file to extract shape data.
- void freeShapeData (ShapeData ∗data, int count)

  Frees the dynamically allocated memory for an array of ShapeData.
- int getExpectedFeatureCount (const char ∗filename)

  Determines the expected number of features in a file based on its extension.
- int parseFilename (const char ∗filename, int ∗class, int ∗sample)

  Parses the filename to extract class and sample information.
- int allocateFeatures (ShapeData ∗data)

  Allocates memory for storing feature values in ShapeData.
- int readFeaturesFromFile (FILE ∗file, double ∗features, int featureCount)

  Reads feature values from a file and stores them in an array.

4.3.1 Detailed Description

Header file for reading and processing shape data from files.

4.3.2 Function Documentation

4.3.2.1 allocateFeatures()

```
int allocateFeatures (
            ShapeData * data )
```

Allocates memory for storing feature values in ShapeData.

This function allocates memory for storing feature values in a ShapeData structure.

Parameters

data	Pointer to the ShapeData structure where features should be stored.

Returns

SUCCESS if memory allocation is successful, ERR_MEMORY_ALLOCATION_FAILED otherwise.

4.3.2.2 freeShapeData()

```
void freeShapeData (
            ShapeData * data,
            int count )
```

Frees the dynamically allocated memory for an array of ShapeData.

This function frees the memory allocated for an array of ShapeData structures.

Parameters

data	Pointer to the array of ShapeData.
count	Number of elements in the array.

4.3.2.3 getExpectedFeatureCount()

```
int getExpectedFeatureCount (
            const char * filename )
```

Determines the expected number of features in a file based on its extension.

This function determines the expected number of features in a file based on its extension.

Parameters

filename	Path to the file.

Returns

Expected number of features for known file types, or ERR_UNKNOWN_FILE_TYPE for unknown types.

4.3.2.4 parseFilename()

```
int parseFilename (
            const char * filename,
            int * class,
            int * sample )
```

Parses the filename to extract class and sample information.

This function parses the filename to extract class and sample information.

Parameters

filename	The filename to parse.
class	Pointer to store the extracted class number.
sample	Pointer to store the extracted sample number.

Returns

SUCCESS if parsing is successful, ERR_INVALID_FILENAME otherwise.

4.3.2.5 readAllFiles()

```
ShapeData* readAllFiles (
            const char * directory,
            const char * extension,
            int * count )
```

Reads all files with a specified extension in a given directory.

This function reads all files with the specified extension in a directory, extracts shape data from each file, and returns an array of ShapeData structures.

Parameters

directory	Path to the directory containing files.
extension	File extension to filter the files to be read.
count	Pointer to an integer to store the number of files read.

Returns

Pointer to an array of ShapeData, each element representing one file's data.

4.3.2.6 readFeaturesFromFile()

```
int readFeaturesFromFile (
            FILE * file,
            double * features,
            int featureCount )
```

Reads feature values from a file and stores them in an array.

This function reads feature values from a file and stores them in an array.

Parameters

file	Pointer to the FILE object to read from.
features	Pointer to the array where features should be stored.
featureCount	The number of features to read.

Returns

SUCCESS if all features are read successfully, ERR_FEATURES_VALUES otherwise.

4.3.2.7 readFile()

```
ShapeData readFile (
            const char * filename )
```

Reads and processes a single file to extract shape data.

This function reads a single file, extracts class, sample, and feature information, and returns a ShapeData structure representing the data from the file.

Parameters

filename	Path to the file to be read.

Returns

ShapeData structure containing data extracted from the file.

4.4 data_split.h File Reference

Header file for functions related to splitting shape data into training and test sets.

```
#include "data_reader.h"
#include <time.h>
```

Include dependency graph for data_split.h: This graph shows which files directly or indirectly include this file:

Classes

- struct SplitData

Macros

- #define **SPLIT_SUCCESS** 0
- #define **SPLIT_ERR_INVALID_INPUT** -1
- #define **SPLIT_ERR_MEMORY_FAILURE** -2

Functions

- SplitData splitData (ShapeData *shapes, int totalSize, float trainingFraction)

  Splits shape data into training and test sets.
- void shuffleData (ShapeData *shapes, int size)

  Shuffles an array of ShapeData.
- void freeSplitData (SplitData *split)

  Frees the dynamically allocated memory in a SplitData structure.

4.4.1 Detailed Description

Header file for functions related to splitting shape data into training and test sets.

4.4.2 Function Documentation

4.4.2.1 freeSplitData()

```
void freeSplitData (
            SplitData * split )
```

Frees the dynamically allocated memory in a SplitData structure.

Parameters

split	Pointer to the SplitData structure to be freed

4.4.2.2 shuffleData()

```
void shuffleData (
            ShapeData * shapes,
            int size )
```

Shuffles an array of ShapeData.

Algorithm for randomizing the order of elements.

Parameters

shapes	Pointer to the array of ShapeData to shuffle
size	Number of elements in the shapes array

4.4.2.3 splitData()

```
SplitData splitData (
            ShapeData * shapes,
            int totalSize,
            float trainingFraction )
```

Splits shape data into training and test sets.

The function shuffles the data before splitting to ensure random distribution.

Parameters

shapes	Pointer to the array of ShapeData to be split
totalSize	Total number of elements in the shapes array
trainingFraction	Fraction of data to be used for training (0.0 - 1.0)

Returns

A SplitData structure containing training and test sets

4.5 kmeans_evaluation.h File Reference

```
#include "kmeans.h"
#include "knn.h"
```

Include dependency graph for kmeans_evaluation.h:

Functions

- double silhouetteScore (Cluster *clusters, int k, int featureCount)

  Calculates the silhouette score for clustering.
- double withinClusterSumOfSquares (Cluster *clusters, int k, int featureCount)

  Computes the within-cluster sum of squares.
- double betweenClusterSumOfSquares (Cluster *clusters, int k, int featureCount, ShapeData *globalCentroid, int dataSize)

  Calculates the between-cluster sum of squares.
- ShapeData calculateGlobalCentroid (ShapeData *shapes, int count, int featureCount)

  Calculates the global centroid of all data points.

4.5.1 Detailed Description

Header file for K-means clustering evaluation functions.

4.5.2 Function Documentation

4.5.2.1 betweenClusterSumOfSquares()

```
double betweenClusterSumOfSquares (
            Cluster * clusters,
            int k,
            int featureCount,
            ShapeData * globalCentroid,
            int dataSize )
```

Calculates the between-cluster sum of squares.

Parameters

clusters	Pointer to an array of Cluster objects.
k	Number of clusters.
featureCount	Number of features in each data point.
globalCentroid	Pointer to the global centroid.
dataSize	Total number of data points.

Returns

double The total between-cluster sum of squares.

4.5.2.2 calculateGlobalCentroid()

```
ShapeData calculateGlobalCentroid (
            ShapeData * shapes,
```

```
            int count,
            int featureCount )
```

Calculates the global centroid of all data points.

Parameters

shapes	Pointer to an array of ShapeData objects.
count	Number of data points.
featureCount	Number of features in each data point.

Returns

ShapeData The global centroid of all data points.

4.5.2.3 silhouetteScore()

```
double silhouetteScore (
            Cluster * clusters,
            int k,
            int featureCount )
```

Calculates the silhouette score for clustering.

Parameters

clusters	Pointer to an array of Cluster objects.
k	Number of clusters.
featureCount	Number of features in each data point.

Returns

double The average silhouette score of all clusters.

4.5.2.4 withinClusterSumOfSquares()

```
double withinClusterSumOfSquares (
            Cluster * clusters,
            int k,
            int featureCount )
```

Computes the within-cluster sum of squares.

Parameters

clusters	Pointer to an array of Cluster objects.
k	Number of clusters.
featureCount	Number of features in each data point.

Returns

double The total within-cluster sum of squares.

4.6 normalization.h File Reference

Header file for normalizing feature values in ShapeData.

```
#include "data_reader.h"
#include <pthread.h>
```

Include dependency graph for normalization.h:

Classes

- struct ThreadArgs

Functions

- void normalizeData (ShapeData *data, int dataSize, int featureCount)

  Normalizes the feature values in an array of ShapeData.

4.6.1 Detailed Description

Header file for normalizing feature values in ShapeData.

4.6.2 Function Documentation

4.6.2.1 normalizeData()

```
void normalizeData (
            ShapeData * data,
            int dataSize,
            int featureCount )
```

Normalizes the feature values in an array of ShapeData.

Normalization scales the data to a specific range, typically [0, 1].

Parameters

data	Pointer to the array of ShapeData.
dataSize	Total number of ShapeData items.
featureCount	Number of features in each ShapeData item.

4.7 standardization.h File Reference

Header file for standardizing feature values in ShapeData.

```
#include "data_reader.h"
```

Include dependency graph for standardization.h:

Functions

- void standardizeData (ShapeData *data, int dataSize, int featureCount)

  Standardizes the feature values in an array of ShapeData.

4.7.1 Detailed Description

Header file for standardizing feature values in ShapeData.

4.7.2 Function Documentation

4.7.2.1 standardizeData()

```
void standardizeData (
            ShapeData * data,
            int dataSize,
            int featureCount )
```

Standardizes the feature values in an array of ShapeData.

Standardization (or Z-score normalization) scales the data to have a mean of 0 and standard deviation of 1.

Parameters

data	Pointer to the array of ShapeData.
dataSize	Total number of ShapeData items.
featureCount	Number of features in each ShapeData item.

Index