

Introduction à la reconnaissance des formes

Generated by Doxygen 1.9.1

1 Class Index	1
1.1 Class List	1
2 File Index	3
2.1 File List	3
3 Class Documentation	5
3.1 ClassMetrics Struct Reference	5
3.1.1 Detailed Description	5
3.1.2 Member Data Documentation	5
3.1.2.1 accuracy	5
3.1.2.2 f1Score	5
3.1.2.3 precision	6
3.1.2.4 recall	6
3.2 Cluster Struct Reference	6
3.2.1 Detailed Description	6
3.2.2 Member Data Documentation	6
3.2.2.1 centroid	6
3.2.2.2 clusterClass	7
3.2.2.3 points	7
3.2.2.4 size	7
3.3 CommandLineOptions Struct Reference	7
3.3.1 Detailed Description	7
3.3.2 Member Data Documentation	7
3.3.2.1 directory	8
3.3.2.2 extension	8
3.3.2.3 k	8
3.3.2.4 method	8
3.3.2.5 p	8
3.3.2.6 preprocessing	8
3.3.2.7 trainingFraction	8
3.4 ConfusionMatrix Struct Reference	9
3.4.1 Detailed Description	9
3.4.2 Member Data Documentation	9
3.4.2.1 classCount	9
3.4.2.2 matrix	9
3.5 ConfusionMatrixMetrics Struct Reference	9
3.5.1 Detailed Description	10
3.5.2 Member Data Documentation	10
3.5.2.1 classCount	10
3.5.2.2 classMetrics	10
3.5.2.3 overallMetrics	10
3.6 CrossValidationMetrics Struct Reference	10

3.6.1 Detailed Description	11
3.6.2 Member Data Documentation	11
3.6.2.1 foldsProcessed	11
3.6.2.2 totalAccuracy	11
3.6.2.3 totalF1Score	11
3.6.2.4 totalPrecision	11
3.6.2.5 totalRecall	11
3.7 DistanceLabel Struct Reference	12
3.7.1 Detailed Description	12
3.7.2 Member Data Documentation	12
3.7.2.1 distance	12
3.7.2.2 label	12
3.8 ShapeData Struct Reference	12
3.8.1 Detailed Description	13
3.8.2 Member Data Documentation	13
3.8.2.1 class	13
3.8.2.2 featureCount	13
3.8.2.3 features	13
3.8.2.4 sample	13
3.9 SplitData Struct Reference	13
3.9.1 Member Data Documentation	14
3.9.1.1 testSet	14
3.9.1.2 testSize	14
3.9.1.3 trainingSet	14
3.9.1.4 trainingSize	14
3.10 ThreadArgs Struct Reference	14
3.10.1 Member Data Documentation	15
3.10.1.1 data	15
3.10.1.2 endIdx	15
3.10.1.3 featureCount	15
3.10.1.4 max	15
3.10.1.5 min	15
3.10.1.6 startIdx	15
4 File Documentation	17
4.1 confusion_matrix.h File Reference	17
4.2 cross_validation.h File Reference	17
4.2.1 Detailed Description	18
4.2.2 Typedef Documentation	18
4.2.2.1 ModelFunction	18
4.2.3 Function Documentation	18
4.2.3.1 crossValidation()	18

4.2.3.2 knnModelFunction()	19
4.3 data_reader.h File Reference	19
4.3.1 Detailed Description	20
4.3.2 Function Documentation	20
4.3.2.1 allocateFeatures()	20
4.3.2.2 freeShapeData()	20
4.3.2.3 getExpectedFeatureCount()	20
4.3.2.4 parseFilename()	21
4.3.2.5 readAllFiles()	21
4.3.2.6 readFeaturesFromFile()	22
4.3.2.7 readFile()	22
4.4 data_split.h File Reference	23
4.4.1 Detailed Description	23
4.4.2 Function Documentation	23
4.4.2.1 freeSplitData()	23
4.4.2.2 shuffleData()	24
4.4.2.3 splitData()	24
4.5 kmeans_evaluation.h File Reference	24
4.5.1 Detailed Description	25
4.5.2 Function Documentation	25
4.5.2.1 betweenClusterSumOfSquares()	25
4.5.2.2 calculateGlobalCentroid()	25
4.5.2.3 silhouetteScore()	26
4.5.2.4 withinClusterSumOfSquares()	26
4.6 normalization.h File Reference	27
4.6.1 Detailed Description	27
4.6.2 Function Documentation	27
4.6.2.1 normalizeData()	27
4.7 standardization.h File Reference	28
4.7.1 Detailed Description	28
4.7.2 Function Documentation	28
4.7.2.1 standardizeData()	28
Index	29

Chapter 1

Class Index

1.1 Class List

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

ClassMetrics	
Metrics for evaluating classification performance for a single class	5
Cluster	6
CommandLineOptions	
Stores command line options	7
ConfusionMatrix	
Represents a confusion matrix for classification results	9
ConfusionMatrixMetrics	
Container for metrics calculated from a confusion matrix	9
CrossValidationMetrics	
Structure to hold aggregated metrics from k-fold cross-validation	10
DistanceLabel	12
ShapeData	
Structure representing the data of a shape	12
SplitData	13
ThreadArgs	14

Chapter 2

File Index

2.1 File List

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

confusion_matrix.h	
Header file for confusion matrix operations	17
cross_validation.h	
Header file for implementing k-fold cross-validation with model evaluation metrics	17
data_reader.h	
Header file for reading and processing shape data from files	19
data_split.h	
Header file for functions related to splitting shape data into training and test sets	23
kmeans.h	??
kmeans_evaluation.h	24
knn.h	??
normalization.h	
Header file for normalizing feature values in ShapeData	27
standardization.h	
Header file for standardizing feature values in ShapeData	28

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

<i>split</i>	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

<i>data</i>	Pointer to the dataset.
<i>dataSize</i>	Number of elements in the dataset.
<i>kFolds</i>	Number of folds for cross-validation.
<i>modelFunc</i>	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

<i>split</i>	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

<i>data</i>	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

<i>data</i>	Pointer to the array of ShapeData .
<i>count</i>	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

<i>filename</i>	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

<i>filename</i>	The filename to parse.
<i>class</i>	Pointer to store the extracted class number.
<i>sample</i>	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

<i>directory</i>	Path to the directory containing files.
<i>extension</i>	File extension to filter the files to be read.
<i>count</i>	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

<i>file</i>	Pointer to the FILE object to read from.
<i>features</i>	Pointer to the array where features should be stored.
<i>featureCount</i>	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

<i>filename</i>	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

<i>split</i>	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

<i>shapes</i>	Pointer to the array of ShapeData to shuffle
<i>size</i>	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

<i>shapes</i>	Pointer to the array of ShapeData to be split
<i>totalSize</i>	Total number of elements in the shapes array
<i>trainingFraction</i>	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

<i>clusters</i>	Pointer to an array of Cluster objects.
<i>k</i>	Number of clusters.
<i>featureCount</i>	Number of features in each data point.
<i>globalCentroid</i>	Pointer to the global centroid.
<i>dataSize</i>	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

<i>shapes</i>	Pointer to an array of ShapeData objects.
<i>count</i>	Number of data points.
<i>featureCount</i>	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

<i>clusters</i>	Pointer to an array of Cluster objects.
<i>k</i>	Number of clusters.
<i>featureCount</i>	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

<i>clusters</i>	Pointer to an array of Cluster objects.
<i>k</i>	Number of clusters.
<i>featureCount</i>	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

<i>data</i>	Pointer to the array of ShapeData .
<i>dataSize</i>	Total number of ShapeData items.
<i>featureCount</i>	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

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

Index

- accuracy
 - ClassMetrics, 5
- allocateFeatures
 - data_reader.h, 20
- betweenClusterSumOfSquares
 - kmeans_evaluation.h, 25
- calculateGlobalCentroid
 - kmeans_evaluation.h, 25
- centroid
 - Cluster, 6
- class
 - ShapeData, 13
- classCount
 - ConfusionMatrix, 9
 - ConfusionMatrixMetrics, 10
- ClassMetrics, 5
 - accuracy, 5
 - f1Score, 5
 - precision, 5
 - recall, 6
- classMetrics
 - ConfusionMatrixMetrics, 10
- Cluster, 6
 - centroid, 6
 - clusterClass, 6
 - points, 7
 - size, 7
- clusterClass
 - Cluster, 6
- CommandLineOptions, 7
 - directory, 7
 - extension, 8
 - k, 8
 - method, 8
 - p, 8
 - preprocessing, 8
 - trainingFraction, 8
- confusion_matrix.h, 17
- ConfusionMatrix, 9
 - classCount, 9
 - matrix, 9
- ConfusionMatrixMetrics, 9
 - classCount, 10
 - classMetrics, 10
 - overallMetrics, 10
- cross_validation.h, 17
 - crossValidation, 18
 - knnModelFunction, 18
 - ModelFunction, 18
- crossValidation
 - cross_validation.h, 18
- CrossValidationMetrics, 10
 - foldsProcessed, 11
 - totalAccuracy, 11
 - totalF1Score, 11
 - totalPrecision, 11
 - totalRecall, 11
- data
 - ThreadArgs, 15
- data_reader.h, 19
 - allocateFeatures, 20
 - freeShapeData, 20
 - getExpectedFeatureCount, 20
 - parseFilename, 21
 - readAllFiles, 21
 - readFeaturesFromFile, 22
 - readFile, 22
- data_split.h, 23
 - freeSplitData, 23
 - shuffleData, 24
 - splitData, 24
- directory
 - CommandLineOptions, 7
- distance
 - DistanceLabel, 12
- DistanceLabel, 12
 - distance, 12
 - label, 12
- endIdx
 - ThreadArgs, 15
- extension
 - CommandLineOptions, 8
- f1Score
 - ClassMetrics, 5
- featureCount
 - ShapeData, 13
 - ThreadArgs, 15
- features
 - ShapeData, 13
- foldsProcessed
 - CrossValidationMetrics, 11
- freeShapeData
 - data_reader.h, 20
- freeSplitData
 - data_split.h, 23

- getExpectedFeatureCount
 - data_reader.h, 20
- k
 - CommandLineOptions, 8
- kmeans_evaluation.h, 24
 - betweenClusterSumOfSquares, 25
 - calculateGlobalCentroid, 25
 - silhouetteScore, 26
 - withinClusterSumOfSquares, 26
- knnModelFunction
 - cross_validation.h, 18
- label
 - DistanceLabel, 12
- matrix
 - ConfusionMatrix, 9
- max
 - ThreadArgs, 15
- method
 - CommandLineOptions, 8
- min
 - ThreadArgs, 15
- ModelFunction
 - cross_validation.h, 18
- normalization.h, 27
 - normalizeData, 27
- normalizeData
 - normalization.h, 27
- overallMetrics
 - ConfusionMatrixMetrics, 10
- p
 - CommandLineOptions, 8
- parseFilename
 - data_reader.h, 21
- points
 - Cluster, 7
- precision
 - ClassMetrics, 5
- preprocessing
 - CommandLineOptions, 8
- readAllFiles
 - data_reader.h, 21
- readFeaturesFromFile
 - data_reader.h, 22
- readFile
 - data_reader.h, 22
- recall
 - ClassMetrics, 6
- sample
 - ShapeData, 13
- ShapeData, 12
 - class, 13
 - featureCount, 13
 - features, 13
 - sample, 13
- shuffleData
 - data_split.h, 24
- silhouetteScore
 - kmeans_evaluation.h, 26
- size
 - Cluster, 7
- SplitData, 13
 - testSet, 14
 - testSize, 14
 - trainingSet, 14
 - trainingSize, 14
- splitData
 - data_split.h, 24
- standardization.h, 28
 - standardizeData, 28
- standardizeData
 - standardization.h, 28
- startIdx
 - ThreadArgs, 15
- testSet
 - SplitData, 14
- testSize
 - SplitData, 14
- ThreadArgs, 14
 - data, 15
 - endIdx, 15
 - featureCount, 15
 - max, 15
 - min, 15
 - startIdx, 15
- totalAccuracy
 - CrossValidationMetrics, 11
- totalF1Score
 - CrossValidationMetrics, 11
- totalPrecision
 - CrossValidationMetrics, 11
- totalRecall
 - CrossValidationMetrics, 11
- trainingFraction
 - CommandLineOptions, 8
- trainingSet
 - SplitData, 14
- trainingSize
 - SplitData, 14
- withinClusterSumOfSquares
 - kmeans_evaluation.h, 26