DCoL

1.1

Generated by Doxygen 1.5.5

Fri Dec 10 01:04:57 2010

Contents

Chapter 1

Directory Hierarchy

1.1 Directories

This directory hierarchy is sorted roughly, but not completely, alphabetically:

Chapter 2

Data Structure Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 4

Directory Documentation

4.1 DistanceFunctions/ Directory Reference

Files

- file **DistanceFunction.h**
- file **EuclideanFunction.h**
- file **NormalizedEuclideanFunction.h**
- file **OverlapFunction.h**
- file **StdWeightedEuclideanFunction.h**
- file **VDMFunction.h**

Chapter 5

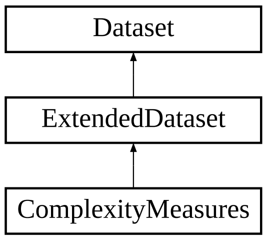
Data Structure Documentation

5.1 ComplexityMeasures Class Reference

Extends from the **ExtendedDataset** (p. 44) class and implements all the complexity measures proposed by Ho and Basu (2002) and some new measures.

`#include <ComplexityMeasures.h>`

Inheritance diagram for ComplexityMeasures::



Public Member Functions

- **ComplexityMeasures** (std::string fileName, bool readAttInfo, bool repUnknownVal=true, int realAttributesFunction=1, int nominalAttributesFunction=1)
- ∼**ComplexityMeasures** ()
- float **computeBoundary** (int ∗∗spanTree)
- int ∗∗ **computePrim** ()
- float **computeIntraInter** ()
- float **computeNonLinearityKNNTrain** (int k=1)
- float **computeNonLinearityKNNConvexHull** (int k=1)
- float **computeNonLinearityLCDistance** (float ∗w, float b)
- float **computeNonLinearityLCTrain** (float ∗w, float b)
- float **computeNonLinearityLCConvexHull** (float ∗w, float b)
- float ∗ **trainSMO** (float &b)
- float **averageNumberOfSamplesPerDimension** ()
- float ∗ **computeFractMaxCoveringSpheres** ()
- float **computeFisher** (int &whichAttribute)
- float **computeFisherVectorized** ()

- float **computeVolumeOverlap** ()
- float * **computeMaximumEfficiencyOfAttributes** (int &mostDiscrAtt, float &discPowerOfTheBest)

Protected Member Functions

- float **computeFisher2Class** (int &whichAttribute)
- float **computeFisherMClass** (int &whichAttribute)

5.1.1 Detailed Description

Extends from the **ExtendedDataset** (p. 44) class and implements all the complexity measures proposed by Ho and Basu (2002) and some new measures.

In summary, this class implements the following measures:

1. Measures of overlaps in the feature values from different classes
   (a) Ratio of the maximum Fisher's discriminant (F1)
   (b) Directional-vector maximum Fisher's discriminant ratio (F1v)
   (c) Overlap of the per-class bounding boxes (F2)
   (d) Maximum individual feature efficiency (F3)
   (e) Collective feature efficiency (F4)
2. Measures of class separability
   (a) Minimized sum of the error distance of a linear classifier (L1)
   (b) Training error of a linear classifier (L2)
   (c) Fraction of points on the class boundary (N1)
   (d) Ratio of average intra/inter class nearest neighbor distance (N2)
   (e) Leave-one-out error rate of the one-nearest neighbor classifier (N3)
3. Measures of geometry, topology, and density of manifolds
   (a) Nonlinearity of a linear classifier (L3)
   (b) Nonlinearity of the one-nearest neighbor classifier (N4)
   (c) Fraction of maximum covering spheres (T1)
   (d) Average number of points per dimension (T2)

The implementation of the methods is organized into two .cpp files:

1. ComplexityFunctions.cpp, which implements all the functions of the complexity measures.
2. SMO.cpp, which implements the sequential minimal optimization to train support vector machines (Platt, 1998). This is used as a linear classifier.

Author:

Albert Orriols-Puig and Nuria Macia
Grup de Recerca en Sistemes Intel.ligents
La Salle - Universitat Ramon Llull
C/ Quatre Camins, 2. 08022, Barcelona (Spain)

Date:

Created April, 2009
Last modified December, 2010

5.1.2 Constructor & Destructor Documentation

5.1.2.1 ComplexityMeasures::ComplexityMeasures (std::string *fileName*, bool *readAttInfo*, bool *repUnknownVal* = `true`, int *realAttributesFunction* = `1`, int *nominalAttributesFunction* = `1`) `[inline]`

Constructs a new **ComplexityMeasures** (p. 9) object.

It calls the constructor of the class **Dataset** (p. 17).

Parameters:

fileName is the name of the KEEL/WEKA formatted input file.

readAttInfo indicates whether the information of the attributes has to be read.

repUnknownVal indicates whether the unknown values have to be replaced.

realAttributesFunction indicates the type of distance function for continuous attributes.

nominalAttributesFunction indicates the type of distance function for nominal attributes.

5.1.2.2 ComplexityMeasures::~ComplexityMeasures () `[inline]`

Destructs the **ComplexityMeasures** (p. 9) object.

5.1.3 Member Function Documentation

5.1.3.1 float ComplexityMeasures::computeBoundary (int ** *spanTree*)

Computes the fraction of points on the class boundary (N1).

This measure provides an estimate of the length of the class boundary.

Parameters:

spanTree is the minimum spanning tree calculated by Prim's algorithm.

Returns:

A float with the value of the measure N1.

See also:

computePrim (p. 12).

5.1.3.2 int** ComplexityMeasures::computePrim ()

Runs the Prim's algorithm on the complete graph represented by the distances between examples.

Returns:

An integer* with N-1 pairs of nearest neighbors, being N the number of examples.

See also:

computeBoundary (p. 11).

5.1.3.3 float ComplexityMeasures::computeIntraInter ()

Computes the ratio of average intra/inter class nearest neighbor distance (N2).

This measure calculates, for each example, the distance to its nearest neighbor of the same class and of another class. Then, the ratio between the intra-class distance and the inter-class distance is returned.

Returns:

A float with the value of the measure N2.

5.1.3.4 float ComplexityMeasures::computeNonLinearityKNNTrain (int *k* = 1)

Computes the leave-one-out error rate of the one-nearest neighbor classifier (N3).

Parameters:

k is the number of neighbors to be used (default 1) for the kNN algorithm.

Returns:

A float with the value of the measure N3.

See also:

createExamplesByInterpolation.

5.1.3.5 float ComplexityMeasures::computeNonLinearityKNNConvexHull (int *k* = 1)

Computes the nonlinearity of a linear classifier (L3).

This measure uses the kNN learning algorithm on an artificially generated data set on the convex hull between classes.

Parameters:

k is the number of neighbors to be used (default 1) for the kNN algorithm.

Returns:

A float with the value of the measure L3.

See also:

createExamplesByInterpolation.

5.1.3.6 float ComplexityMeasures::computeNonLinearityLCDistance (float ∗ *w*, float *b*)

Computes the sum of the error distance of a linear classifier (L1).

Parameters:

w is a vector of weights that represent the SVM.

b is the offset of the SVM.

Returns:

A float with the value of the measure L1.

See also:

createExamplesByInterpolation.

5.1.3.7 float ComplexityMeasures::computeNonLinearityLCTrain (float ∗ *w*, float *b*)

Computes the training error of a linear classifier (L2).

Parameters:

w is a vector of weights that represent the SVM.

b is the offset of the SVM.

Returns:

A float with the value of the measure L2.

See also:

createExamplesByInterpolation.

5.1.3.8 float ComplexityMeasures::computeNonLinearityLCConvexHull (float ∗ *w*, float *b*)

Computes the nonlinearity of the one-nearest neighbor classifier (N4).

The error rate is measured on examples artificially generated on the convex hull between classes.

Parameters:

w is a vector of weights that represent the SVM.

b is the offset of the SVM.

Returns:

A float with the value of the measure N4.

See also:

createExamplesByInterpolation.

5.1.3.9 float∗ ComplexityMeasures::trainSMO (float & *b*)

Trains a support vector machine by means of the sequential minimal optimization algorithm (Platt, 1998).

Parameters:

b is a reference to the offset of the SVM.

Returns:

A float∗ with a vector of weights that define the linear kernel.

5.1.3.10 float ComplexityMeasures::averageNumberOfSamplesPerDimension ()

Computes the average number of samples per dimension (T2).

Returns:

A float with the value of the measure T2.

5.1.3.11 float∗ ComplexityMeasures::computeFractMaxCoveringSpheres ()

Computes the fraction of maximum covering spheres (T1).

Returns:

A float with the value of the measure T1.

5.1.3.12 float ComplexityMeasures::computeFisher (int & *whichAttribute*)

Computes the Fisher's discriminant ratio (F1) according to its directional-vector definition.

Parameters:

whichAttribute indicates the attribute that maximally discriminates.

Returns:

A float with the value of the measure F1.

5.1.3.13 float ComplexityMeasures::computeFisherVectorized ()

Computes the Fisher's discriminant ratio projected over the vector with the best direction (F1v).

Returns:

A float with the measure applied to the most discriminant feature.

5.1.3.14 float ComplexityMeasures::computeVolumeOverlap ()

Computes the overlap of the per-class bounding boxes for data sets with any number of classes.

We extended the initial definition of the measure provided in (Ho and Basu, 2002) for two-class data sets to m-class data sets as recommended in (Ho, Basu, and Law, 2006).

Returns:

A float with the measure applied to the most discriminant feature.

5.1.3.15 float∗ ComplexityMeasures::computeMaximumEfficiencyOfAttributes (int & *mostDiscrAtt*, float & *discPowerOfTheBest*)

Computes the maximum efficiency of all the attributes (F3,F4).

It computes the discriminatory capabilities of each attribute.

Parameters:

mostDiscrAtt contains the position of the most discriminative attribute is returned (it is passed by reference).

discPowerOfTheBest contains the discriminant power of the most discriminative attribute is returned (it is passed by reference).

Returns:

A float∗ with the value of the measure F3 and F4.

5.1.3.16 float ComplexityMeasures::computeFisher2Class (int & *whichAttribute*) `[protected]`

Computes the Fisher's discriminant ratio for two-class problems (F1).

Parameters:

whichAttribute indicates the attribute that maximally discriminates.

Returns:

A float with the value of the measure F1.

5.1.3.17 float ComplexityMeasures::computeFisherMClass (int & *whichAttribute*) `[protected]`

Computes the Fisher's discriminant ratio for m-class problems (m>2) (F1).

Parameters:

whichAttribute indicates the attribute that maximally discriminates.

Returns:

A float with the value of the measure F1.

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

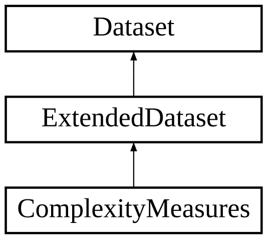
- ComplexityMeasures.h

5.2 Dataset Class Reference

Implements methods to read, manipulate, and print data.

`#include <Dataset.h>`

Inheritance diagram for Dataset::



Public Member Functions

- **Dataset** (std::string fileName, bool readAttInfo, bool repUnknownVal=true)
- ~**Dataset** ()
- float * **getRandomExample** (int &wClass)
- float * **getCurrentExample** ()
- int **getCurrentClass** ()
- void **beginSequentialExamples** ()
- float * **getNextExample** (int &wClass)
- int **getNumberOfExamples** ()
- int **getNumberOfAttributes** ()
- int **getNumberOfClasses** ()
- void **makeInitialStatistics** ()
- std::string **getName** ()
- int **getClassOfExample** (int i)

Static Public Attributes

- static float **UNKNOWN_VALUE**

Protected Member Functions

- void **initConfigParameters** (bool readAttInfo)
- void **initAttributes** ()
- void **initParameters** (bool readAttInfo)
- void **reserveMemoryForStatistics** ()
- void **readData** (std::ifstream &fin, bool readAttInfo)
- void **getBasicInfo** (std::string &fileName, bool readAttInfo)
- void **getBasicInfo** (std::string &fileName, bool readAttInfo, int &_attNum, int &_exNum)
- void **readAttributes** (std::ifstream &fin, bool readAttInfo, bool readStatistics=false)
- void **readClassInformation** (std::string &s)
- void **readAttributeInformation** (std::string &s, **StringTokenizer** &st, int attCount)
- std::string **toLowerCase** (std::string s)
- void **transformDateAttributesToNumeric** ()

- void **transformDateAttributeToNumeric** (int whichAttribute)
- float **convertToOrdinal** (**Date** value, **Date** *convertionArray, int arrayLength)
- void **normalizeContinuousAttributes** ()
- void **normalizeNominalAttributes** ()
- void **normalizeAllAttributes** ()
- void **unNormalizeAllAttributes** ()
- void **replaceUnknownValues** ()
- int **getIntegerFromVector** (std::string **attName**, std::string *vect, int max, std::string s)
- void **calculateAverages** ()
- void **calculateDeviations** ()
- void **removeAllZeroExamples** ()
- **Matrix** * **getMeanVectorOfClass** (int wClass)
- **Matrix** * **getCovarianceMatrixOfClass** (int wClass)

Protected Attributes

- float ** **example**
- int * **classOfExample**
- int **currentExample**
- int **numberOfAttributes**
- int **numberOfExamples**
- int **sequentialExamples**
- int **numberOfClasses**
- std::string **relationName**
- std::string * **attName**
- char * **typeOfRepresentation**
- int **numberOfContinuousAttributes**
- int **numberOfNominalAttributes**
- int * **nominalAttrNumValues**
- std::string ** **enumAtt**
- float * **minAttValue**
- float * **maxAttValue**
- float * **avgAttValue**
- int * **numberValuedAtt**
- float * **stdAttValue**
- float ** **minAttValuePerClass**
- float ** **maxAttValuePerClass**
- std::string **className**
- int **classPosition**
- char **classType**
- std::string * **nominalClassValues**
- int **minClassValue**
- int **maxClassValue**
- float *** **avgNominalValue**
- float ** **avgRealValue**
- float ** **stdRealValue**
- float ** **avgAttCounter**
- bool **areDataNormalized**
- bool **areContinuousAttributesNormalized**
- bool **areNominalAttributesNormalized**
- bool **areUnknownValuesReplaced**
- **DateContainer** ** **dateFormat**
- **Date** ** **dateContent**

5.2.1 Detailed Description

Implements methods to read, manipulate, and print data.

This class reads the information from the data set (in KEEL/WEKA format) and provides methods to deal with data.

The implementation of the methods is organized into two .cpp files:

1. Dataset.cpp, which contains the methods to read, write, and deal with the data.
2. Statistics.cpp, which contains the methods to make statistics.

Author:

Albert Orriols-Puig and Nuria Macia
Grup de Recerca en Sistemes Intel.ligents
La Salle - Universitat Ramon Llull
C/ Quatre Camins, 2. 08022, Barcelona (Spain)

Date:

Created April, 2009
Last modified December, 2010

5.2.2 Constructor & Destructor Documentation

5.2.2.1 Dataset::Dataset (std::string *fileName*, bool *readAttInfo*, bool *repUnknownVal* = `true`)

Constructs a new **Dataset** (p. 17) object: opens the file and the examples in there.

Parameters:

fileName is the name of the KEEL/WEKA formatted input file.
readAttInfo indicates whether the information of the attributes has to be read.
repUnknownVal indicates whether the unknown values need to be replaced.

5.2.2.2 Dataset::∼Dataset ()

Destructs the **Dataset** (p. 17) object.

5.2.3 Member Function Documentation

5.2.3.1 float* Dataset::getRandomExample (int & *wClass*)

Returns a randomly chosen example.

Parameters:

wClass is the class of the example (it is passed by reference).

Returns:

A float* with the example.

5.2.3.2 float* Dataset::getCurrentExample ()

Returns the current example.

Returns:

A float* with the current example.

5.2.3.3 int Dataset::getCurrentClass ()

Returns the class of the current example.

Returns:

An integer with the class of the current example.

5.2.3.4 void Dataset::beginSequentialExamples ()

Indicates that the examples will be explored sequentially.

5.2.3.5 float* Dataset::getNextExample (int & *wClass*)

Returns the next example (sequential exploration).

Parameters:

wClass is a reference where the class of the example is returned.

Returns:

A float* with the next example.

5.2.3.6 int Dataset::getNumberOfExamples ()

Returns the number of examples of the data set.

Returns:

An integer with the number of examples.

5.2.3.7 int Dataset::getNumberOfAttributes ()

Returns the number of attributes of the data set.

Returns:

An integer with the number of attributes.

5.2.3.8 int Dataset::getNumberOfClasses ()

Returns the number of classes of the data set.

Returns:

An integer with the number of classes.

5.2.3.9 void Dataset::makeInitialStatistics ()

Computes the initial statistics on the data.

5.2.3.10 std::string Dataset::getName () `[inline]`

Returns the name of the data set.

Returns:

A string with the relationship name.

References relationName.

5.2.3.11 int Dataset::getClassOfExample (int *i*) `[inline]`

Returns the class of the ith example.

Returns:

An integer with the class value of the ith example.

References classOfExample.

5.2.3.12 void Dataset::initConfigParameters (bool *readAttInfo*) `[protected]`

Initializes the configuration parameters.

Parameters:

readAttInfo indicates whether the information of the attributes has to be read.

5.2.3.13 void Dataset::initAttributes () `[protected]`

Initializes the attributes.

5.2.3.14 void Dataset::initParameters (bool *readAttInfo*) `[protected]`

Initializes several internal parameters.

Parameters:

readAttInfo indicates whether the information of the attributes has to be read.

5.2.3.15 void Dataset::reserveMemoryForStatistics () `[protected]`

Reserves memory for statistics.

5.2.3.16 void Dataset::readData (std::ifstream & *fin*, bool *readAttInfo*) `[protected]`

Reads the data from file.

Parameters:

fin is the input file.

readAttInfo indicates whether the information of the attributes has to be read.

5.2.3.17 void Dataset::getBasicInfo (std::string & *fileName*, bool *readAttInfo*) `[protected]`

Returns the basic information from the file.

Parameters:

fileName is the name of the file that contains the input data set.

readAttInfo indicates whether the information of the attributes has to be read.

5.2.3.18 void Dataset::getBasicInfo (std::string & *fileName*, bool *readAttInfo*, int & *_attNum*, int & *_exNum*) `[protected]`

Returns the basic information from the file.

Parameters:

fileName is the name of the file that contains the input data set.

readAttInfo indicates whether the information of the attributes has to be read.

_attNum is the number of attributes (it is passed by reference).

_exNum is the number of examples (it is passed by reference).

5.2.3.19 void Dataset::readAttributes (std::ifstream & *fin*, bool *readAttInfo*, bool *readStatistics* = `false`) `[protected]`

Reads the information of the data attributes.

Parameters:

fin is the input file.

readAttInfo indicates whether the information of the attributes has to be read.

readStatistics indicates whether statistics need to be taken after reading the data.

5.2.3.20 void Dataset::readClassInformation (std::string & *s*) `[protected]`

Reads and stores the information of the class.

Parameters:

s is the class that has been read.

5.2.3.21 void Dataset::readAttributeInformation (std::string & *s*, StringTokenizer & *st*, int *attCount*) `[protected]`

Reads and stores the information of a single attribute.

Parameters:

s is the attribute that has been read.

st is the tokenized string.

attCount is the number of the current attribute.

5.2.3.22 std::string Dataset::toLowerCase (std::string *s*) `[protected]`

Transforms the string into lower case.

Parameters:

s is the string that will be transformed into lower case.

5.2.3.23 void Dataset::transformDateAttributesToNumeric () `[protected]`

Transforms all date attributes in the data set into integer attributes.

5.2.3.24 void Dataset::transformDateAttributeToNumeric (int *whichAttribute*) `[protected]`

Transforms the date attribute specified in the input parameter into an integer format.

Parameters:

whichAttribute is the attribute that will be transformed into an integer.

5.2.3.25 float Dataset::convertToOrdinal (Date *value*, Date ∗ *convertionArray*, int *arrayLength*) `[protected]`

Transforms the value passed as parameter to an ordinal value.

Parameters:

value is the value to be transformed.

convertionArray is the array that containes the sorted float values.

arrayLength is the length of the convertionArray array.

Returns:

A float with the converted value.

5.2.3.26 void Dataset::normalizeContinuousAttributes () `[protected]`

Normalizes continuous attributes.

5.2.3.27 void Dataset::normalizeNominalAttributes () `[protected]`

Normalizes nominal attributes.

5.2.3.28 void Dataset::normalizeAllAttributes () `[protected]`

Normalizes all the attributes of the data set.

5.2.3.29 void Dataset::unNormalizeAllAttributes () `[protected]`

Unnormalizes all the attributes of the data set.

5.2.3.30 void Dataset::replaceUnknownValues () `[protected]`

Replaces unknown values.

5.2.3.31 int Dataset::getIntegerFromVector (std::string *attName*, std::string ∗ *vect*, int *max*, std::string *s*) `[protected]`

Transforms the nominal value into its corresponding integer.

Parameters:

attName is the name of the attribute.

vect is the vector with all the possible nominal values for a given attribute.

max is the maximum size of the vector.

s contains the categorical value that has to be transformed into integer.

5.2.3.32 void Dataset::calculateAverages () `[protected]`

Calculates the average values for each attribute.

5.2.3.33 void Dataset::calculateDeviations () `[protected]`

Calculates the deviations for each attribute.

5.2.3.34 void Dataset::removeAllZeroExamples () `[protected]`

Removes unnecessary examples.

5.2.3.35 Matrix* Dataset::getMeanVectorOfClass (int *wClass*) `[protected]`

Returns an n x 1 matrix with the means of the class passed as parameter.

Parameters:

wClass is the class.

Returns:

A matrix with the means per class.

5.2.3.36 Matrix* Dataset::getCovarianceMatrixOfClass (int *wClass*) `[protected]`

Returns an n x n matrix with the covariances of the class passed as parameter.

Parameters:

wClass is the class.

Returns:

A matrix with the covariances per class.

5.2.4 Field Documentation

5.2.4.1 float Dataset::UNKNOWN_VALUE `[static]`

Representation of the unknown value in an example.

5.2.4.2 float** Dataset::example `[protected]`

Data set examples.

5.2.4.3 int* Dataset::classOfExample `[protected]`

Class or label of each example.

Referenced by getClassOfExample().

5.2.4.4 int Dataset::currentExample `[protected]`

Index of the current example.

5.2.4.5 int Dataset::numberOfAttributes `[protected]`

Number of input attributes.

The class attribute is not considered.

5.2.4.6 int Dataset::numberOfExamples `[protected]`

Number of examples.

5.2.4.7 int Dataset::sequentialExamples `[protected]`

Shows whether the examples are sampled sequentially.

5.2.4.8 int Dataset::numberOfClasses `[protected]`

Number of different classes.

5.2.4.9 std::string Dataset::relationName `[protected]`

Name of the data set.

Referenced by getName().

5.2.4.10 std::string* Dataset::attName `[protected]`

Array with the name of the attributes.

5.2.4.11 char* Dataset::typeOfRepresentation `[protected]`

Array with the type of each input attribute.

5.2.4.12 int Dataset::numberOfContinuousAttributes `[protected]`

Number of continuous attributes.

5.2.4.13 int Dataset::numberOfNominalAttributes `[protected]`

Number of nominal attributes.

5.2.4.14 int* Dataset::nominalAttrNumValues `[protected]`

Array with the number of values that a nominal attribute can take.

5.2.4.15 std::string** Dataset::enumAtt `[protected]`

Array with the possible values of each nominal attribute.

5.2.4.16 float* Dataset::minAttValue `[protected]`

Minimum value for each attribute.

5.2.4.17 float* Dataset::maxAttValue `[protected]`

Maximum value for each attribute.

5.2.4.18 float* Dataset::avgAttValue `[protected]`

Average attribute value.

5.2.4.19 int* Dataset::numberValuedAtt `[protected]`

Number of values for the attribute.

5.2.4.20 float* Dataset::stdAttValue `[protected]`

Standard deviation per attribute.

5.2.4.21 float** Dataset::minAttValuePerClass `[protected]`

Minimum value for each attribute and class (index minAttValuePerClass[attribute][class]).

5.2.4.22 float** Dataset::maxAttValuePerClass `[protected]`

Maximum value for each attribute and class (index maxAttValuePerClass[attribute][class]).

5.2.4.23 std::string Dataset::className `[protected]`

Name of the class attribute.

5.2.4.24 int Dataset::classPosition `[protected]`

Position of the class in the list of attributes.

5.2.4.25 char Dataset::classType `[protected]`

Type of the class.

5.2.4.26 std::string* Dataset::nominalClassValues `[protected]`

Possible values that the class can take if it is nominal.

5.2.4.27 int Dataset::minClassValue `[protected]`

Minimum value of the class.

5.2.4.28 int Dataset::maxClassValue `[protected]`

Maximum value of the class.

5.2.4.29 float*** Dataset::avgNominalValue `[protected]`

Frequency of the values for nominal attributes (index avgNominalValue[attribute][class][value]).

5.2.4.30 float** Dataset::avgRealValue `[protected]`

Mean values per attribute and class (continuous attributes) or median values per attribute and class (nominal attributes) (index avgRealValue[attriherbute][class]).

5.2.4.31 float** Dataset::stdRealValue `[protected]`

Standard deviation per attribute and class (index stdRealValue[attribute][class]).

5.2.4.32 float** Dataset::avgAttCounter `[protected]`

Counter of number of values per each attribute and class (index avgAttCounter[attribute][class]).

5.2.4.33 bool Dataset::areDataNormalized `[protected]`

Indicates whether data have been normalized.

5.2.4.34 bool Dataset::areContinuousAttributesNormalized `[protected]`

Indicates whether the real and integer attributes have been normalized.

5.2.4.35 bool Dataset::areNominalAttributesNormalized `[protected]`

Indicates whether the nominal attributes have been normalized.

5.2.4.36 bool Dataset::areUnknownValuesReplaced `[protected]`

Indicates whether unknown values have been replaced.

5.2.4.37 DateContainer** Dataset::dateFormat `[protected]`

Contains the date format in case of having date attributes.

5.2.4.38 Date** Dataset::dateContent `[protected]`

Contains the dates read initially.

Later, they are transformed to ordinal values.

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

- Dataset.h

5.3 Date Class Reference

Implements methods to initialize and compare dates.

`#include <Date.h>`

Public Member Functions

- **Date** ()
- **Date** (int _year, int _month, int _day, int _hour, int _minutes, int _seconds)
- ~**Date** ()
- void **init** (int _year, int _month, int _day, int _hour, int _minutes, int _seconds)
- void **init** (**Date** &d)
- bool **operator<** (**Date** &d2)
- bool **operator>** (**Date** &d2)
- bool **operator<=** (**Date** &d2)
- bool **operator>=** (**Date** &d2)
- bool **operator==** (**Date** &d2)
- bool **operator!=** (**Date** &d2)
- **Date** & **operator=** (**Date** &d2)

Friends

- class **DateContainer**
- std::ostream & **operator<<** (std::ostream &os, **Date** &d)

5.3.1 Detailed Description

Implements methods to initialize and compare dates.

This class contains the information of a date.

Author:

Albert Orriols-Puig and Nuria Macia
Grup de Recerca en Sistemes Intel.ligents
La Salle - Universitat Ramon Llull
C/ Quatre Camins, 2. 08022, Barcelona (Spain)

Date:

Created June, 2010
Last modified December, 2010

You should have received a copy of the GNU General Public License along with DCoL. If not, see <`http://www.gnu.org/licenses/`>.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 Date::Date ()

Constructs a new **Date** (p. 30) object.

5.3.2.2 Date::Date (int *_year*, int *_month*, int *_day*, int *_hour*, int *_minutes*, int *_seconds*)

Constructs a new **Date** (p. 30) object with the passed values.

Parameters:

_year is the year.

_month is the month.

_day is the day.

_hour is the hour.

_minutes is the minutes.

_seconds is the seconds.

5.3.2.3 Date::~Date () `[inline]`

Destructs the **Date** (p. 30) object.

5.3.3 Member Function Documentation

5.3.3.1 void Date::init (int *_year*, int *_month*, int *_day*, int *_hour*, int *_minutes*, int *_seconds*)

Initializes a **Date** (p. 30) object with the passed values.

Parameters:

_year is the year.

_month is the month.

_day is the day.

_hour is the hour.

_minutes is the minutes.

_seconds is the seconds.

5.3.3.2 void Date::init (Date & *d*)

Initializes a **Date** (p. 30) object with the object.

Parameters:

d is the date to be copied to the class object.

5.3.3.3 bool Date::operator< (Date & *d2*)

Is the "less than" operator.

Parameters:

d2 is the **Date** (p. 30) object with which the current object will be compared to.

Returns:

A Boolean that indicates which object is smaller.

Return values:

true the current object is less than the object d2.

false the current object is greater than or equal to the object d2.

5.3.3.4 bool Date::operator> (Date & *d2*)

Is the "greater than" operator.

Parameters:

d2 is the **Date** (p. 30) object with which the current object will be compared to.

Returns:

A Boolean that indicates which object is greater.

Return values:

true the current object is greater than the object d2.

false the current object is less than or equal to the object d2.

5.3.3.5 bool Date::operator<= (Date & *d2*)

Is the "less than or equal to" operator.

Parameters:

d2 is the **Date** (p. 30) object with which the current object will be compared to.

Returns:

A Boolean that indicates which object is smaller or equal.

Return values:

true the current object is less than or equal to the object d2.

false the current object is greater than the object d2.

5.3.3.6 bool Date::operator>= (Date & *d2*)

Is the "greater than or equal to" operator.

Parameters:

d2 is the **Date** (p. 30) object with which the current object will be compared to.

Returns:

A Boolean that indicates which object is greater or equal.

Return values:

true the current object is greater than or equal to the object d2.

false the current object is less than the object d2.

5.3.3.7 bool Date::operator== (Date & *d2*)

Is the "equal to" operator.

Parameters:

d2 is the **Date** (p. 30) object with which the current object will be compared to.

Returns:

A Boolean that indicates whether the objects are equal.

Return values:

true the current object is equal to the object d2.

false the current object is different from the object d2.

5.3.3.8 bool Date::operator!= (Date & *d2*)

Is the "different from" operator.

Parameters:

d2 is the **Date** (p. 30) object with which the current object will be compared to.

Returns:

A Boolean that indicates whether the objects are different.

Return values:

true the current object is different from the object d2.

false the current object is equal to the object d2.

5.3.3.9 Date& Date::operator= (Date & *d2*)

Is the "copy" operator.

Parameters:

d2 is the **Date** (p. 30) object that will be copied to the current object.

Returns:

A **Date** (p. 30)& that references the copied object.

5.3.4 Friends And Related Function Documentation

5.3.4.1 std::ostream& operator<< (std::ostream & *os*, Date & *d*) `[friend]`

Overloads the standard output.

Parameters:

os is a reference to the standard output.

d is a reference to the date that has to be printed.

Returns:

A reference to ostream.

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

- Date.h

5.4 DateContainer Class Reference

Implements method to create, transform, and delete date strings.

`#include <DateContainer.h>`

Public Member Functions

- **DateContainer** (std::string dateFormat)
- ∼**DateContainer** ()
- float **transformToNumeric** (std::string date)
- long double **transformToLongNumeric** (std::string date)
- void **transformToDate** (std::string date, **Date** &d)

5.4.1 Detailed Description

Implements method to create, transform, and delete date strings.

This class reads dates from strings formatted in a prefixed way.

Author:

Albert Orriols-Puig and Nuria Macia
Grup de Recerca en Sistemes Intel.ligents
La Salle - Universitat Ramon Llull
C/ Quatre Camins, 2. 08022, Barcelona (Spain)

Date:

Created June, 2010
Last modified December, 2010

5.4.2 Constructor & Destructor Documentation

5.4.2.1 DateContainer::DateContainer (std::string *dateFormat*)

Constructs a new **DateContainer** (p. 35) object.

Parameters:

dateFormat specifies the format of the date.

5.4.2.2 DateContainer::~DateContainer () `[inline]`

Destructs the **DateContainer** (p. 35) object.

5.4.3 Member Function Documentation

5.4.3.1 float DateContainer::transformToNumeric (std::string *date*)

Returns a numeric value that corresponds to the date passed as argument.

Parameters:

date is the date to be transformed.

Returns:

A float with the transformed date.

5.4.3.2 long double DateContainer::transformToLongNumeric (std::string *date*)

Returns a numeric value that corresponds to the date passed as argument.

Parameters:

date is the date to be transformed.

Returns:

A long double with the transformed date.

5.4.3.3 void DateContainer::transformToDate (std::string *date*, Date & *d*)

Returns a **Date** (p. 30) object that corresponds to the date passed as argument.

Parameters:

date is the date to be transformed.

d is the object where the date will be written.

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

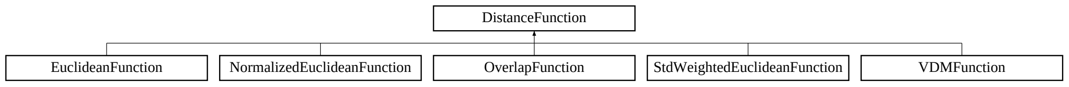
- DateContainer.h

5.5 DistanceFunction Class Reference

Abstract class that defines the distance function classes behavior.

`#include <DistanceFunction.h>`

Inheritance diagram for DistanceFunction::



Public Member Functions

- virtual ~**DistanceFunction** ()
- virtual float **computeDistance** (float att1, float att2)=0

5.5.1 Detailed Description

Abstract class that defines the distance function classes behavior.

This is an abstract class that provides the basic methods of distance functions for individual attributes. Five types of distance functions have been implemented:

1. **EuclideanFunction** (p. 42): Euclidean distance for continuous attributes.
2. **NormalizedEuclideanFunction** (p. 74): Normalized Euclidean distance for continuous attributes.
3. **StdWeightedEuclideanFunction** (p. 81): Standard-deviation weighted Euclidean distance for continuous attributes.
4. **OverlapFunction** (p. 76): Distance for nominal attributes that defines distance 0 for equally valued attributes and distance 1 otherwise.
5. **VDMFunction** (p. 94): Distance between nominal attributes by means of the frequency of each value in the data set.

For further details on these distance functions see (Randall & Martinez, 1997).

Author:

Albert Orriols-Puig and Nuria Macia
Grup de Recerca en Sistemes Intel.ligents
La Salle - Universitat Ramon Llull
C/ Quatre Camins, 2. 08022, Barcelona (Spain)

Date:

Created April, 2009
Last modified December, 2010

This file is part of DCoL.

5.5.2 Constructor & Destructor Documentation

5.5.2.1 virtual DistanceFunction::∼DistanceFunction () `[inline, virtual]`

Destructs the **DistanceFunction** (p. 37).

5.5.3 Member Function Documentation

5.5.3.1 virtual float DistanceFunction::computeDistance (float *att1*, float *att2*) `[pure virtual]`

Computes the distance between two attribute values.

Parameters:

att1 is the value of attribute 1.

att2 is the value of attribute 2.

Returns:

A float with the distance between att1 and att2.

Implemented in **EuclideanFunction** (p. 43), **NormalizedEuclideanFunction** (p. 75), **OverlapFunction** (p. 77), **StdWeightedEuclideanFunction** (p. 82), and **VDMFunction** (p. 95).

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

- DistanceFunctions/DistanceFunction.h

5.6 DistNode Class Reference

Maintains an index and a distance.

`#include <DistNode.h>`

Public Member Functions

- bool **operator<** (**DistNode** &d2)
- bool **operator>** (**DistNode** &d2)
- bool **operator==** (**DistNode** &d2)

Data Fields

- int **index**
- float **dist**

5.6.1 Detailed Description

Maintains an index and a distance.

This class runs the kNN learner, since it permits maintaining the index of the example and its distance to the heap.

Author:

Albert Orriols-Puig and Nuria Macia
Grup de Recerca en Sistemes Intel.ligents
La Salle - Universitat Ramon Llull
C/ Quatre Camins, 2. 08022, Barcelona (Spain)

Date:

Created April, 2009
Last modified December, 2010

5.6.2 Member Function Documentation

5.6.2.1 bool DistNode::operator< (DistNode & *d2*) `[inline]`

Is the "less than" operator.

Parameters:

d2 is the **DistNode** (p. 39) object with which the current object will be compared to.

Returns:

A Boolean that indicates which object is smaller.

Return values:

true the current object is less than the object d2.
false the current object is greater than or equal to the object d2.

References dist.

5.6.2.2 bool DistNode::operator> (DistNode & *d2*) `[inline]`

Is the "greater than" operator.

Parameters:

d2 is the **DistNode** (p. 39) object with which the current object will be compared to.

Returns:

A Boolean that indicates which object is greater.

Return values:

true the current object is greater than the object d2.
false the current object is less than or equal to the object d2.

References dist.

5.6.2.3 bool DistNode::operator== (DistNode & *d2*) `[inline]`

Is the "equal to" operator.

Parameters:

d2 is the **DistNode** (p. 39) object with which the current object will be compared to.

Returns:

A Boolean that indicates whether the objects are equal.

Return values:

true the current object is equal to the object d2.
false the current object is different from the object d2.

References dist.

5.6.3 Field Documentation

5.6.3.1 int DistNode::index

Index of the current node.

5.6.3.2 float DistNode::dist

Distance stored in the current node.

Referenced by operator<(), operator==(), and operator>().

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

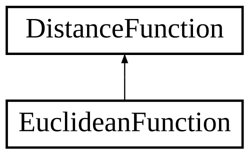
- DistNode.h

5.7 EuclideanFunction Class Reference

Implements the Euclidean distance function for continuous attributes.

`#include <EuclideanFunction.h>`

Inheritance diagram for EuclideanFunction::



Public Member Functions

- **EuclideanFunction** ()
- ~**EuclideanFunction** ()
- float **computeDistance** (float att1, float att2)

5.7.1 Detailed Description

Implements the Euclidean distance function for continuous attributes.

This is an implementation of the **DistanceFunction** (p. 37) abstract class for continuous attributes. It implements the Euclidean distance.

Author:

Albert Orriols-Puig and Nuria Macia
Grup de Recerca en Sistemes Intel.ligents
La Salle - Universitat Ramon Llull
C/ Quatre Camins, 2. 08022, Barcelona (Spain)

Date:

Created April, 2009
Last modified December, 2010

5.7.2 Constructor & Destructor Documentation

5.7.2.1 EuclideanFunction::EuclideanFunction ()

Constructs a new **EuclideanFunction** (p. 42) object.

5.7.2.2 EuclideanFunction::~EuclideanFunction ()

Destructs the **EuclideanFunction** (p. 42) object.

5.7.3 Member Function Documentation

5.7.3.1 float EuclideanFunction::computeDistance (float *att1*, float *att2*) `[virtual]`

Computes the distance between two attribute values.

Parameters:

att1 is the value of attribute 1.

att2 is the value of attribute 2.

Returns:

A float with the distance between att1 and att2.

Implements **DistanceFunction** (p. 38).

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

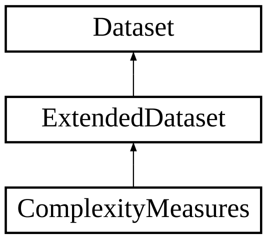
- DistanceFunctions/EuclideanFunction.h

5.8 ExtendedDataset Class Reference

Extends from the **Dataset** (p. 17) class and introduces new functionalities.

`#include <ExtendedDataset.h>`

Inheritance diagram for ExtendedDataset::



Public Member Functions

- **ExtendedDataset** (std::string fileName, bool readAttInfo, bool repUnknownVal=true, int realAttributesFunction=1, int nominalAttributesFunction=1)
- ∼**ExtendedDataset** ()
- void **deleteExamplesPerClass** ()
- float ∗ **getDatasetCharacteristics** ()
- void **organizePerClass** ()
- void **stratifiedCrossValidation** (int k, std::string outName)
- void **crossValidation** (int k, std::string outName)
- std::string ∗ **generate2ClassDatasets** (std::string baseName)
- void **print** (bool printExamples=false)
- std::ofstream & **print** (std::ofstream &fout, bool printNormalizedData)
- std::ofstream & **printOneClassAgainstOthers** (std::ofstream &fout, int whichClass)
- void **instantiateDistanceFunctions** (int realAttributesFunction, int nominalAttributesFunction)

Static Public Attributes

- static float **MAXIMUM_EXAMPLES_FOR_PRECALCULATION**
- static int **EUCLIDEAN**
- static int **NORMALIZED_EUCLIDEAN**
- static int **STD_WEIGHTED_EUCLIDEAN**
- static int **OVERLAP_NOMINAL**
- static int **VDM_NOMINAL**

Protected Member Functions

- void **printExample** (std::ofstream &fout, float ∗inst, int action)
- void **printExampleNoNormalized** (std::ofstream &fout, float ∗inst, int action)
- void **printExample** (float ∗inst, int action)
- void **printHeader** (std::ofstream &fout)
- int **INDEX** (int i, int j)
- float **getDistance** (int i, int j)

- float **getDistance** (float ∗ex1, float ∗ex2)
- float **getApproximateDistance** (int i, int j)
- float **getApproximateDistance** (float ∗ex1, float ∗ex2)

Protected Attributes

- int **typeOfContinuousDistance**
- int **typeOfNominalDistance**
- int ∗ **numberOfExamplesPerClass**
- float ∗∗∗ **examplesPerClass**
- int ∗∗ **indexExamplesPerClass**
- **DistanceFunction** ∗∗ **distanceFunction**
- float ∗ **distances**

5.8.1 Detailed Description

Extends from the **Dataset** (p. 17) class and introduces new functionalities.

The implementation of the methods of data set is done in a single .cpp file: ExtendedDataset.cpp This class includes the following groups of methods:

1. Methods for organizing the examples per class and applying partition procedures.

2. Methods for printing the data set in different formats.

Author:

Albert Orriols-Puig and Nuria Macia
Grup de Recerca en Sistemes Intel.ligents
La Salle - Universitat Ramon Llull
C/ Quatre Camins, 2. 08022, Barcelona (Spain)

Date:

Created April, 2009
Last modified December, 2010

5.8.2 Constructor & Destructor Documentation

5.8.2.1 ExtendedDataset::ExtendedDataset (std::string *fileName*, bool *readAttInfo*, bool *repUnknownVal* = `true`, int *realAttributesFunction* = `1`, int *nominalAttributesFunction* = `1`)

Constructs a new **ExtendedDataset** (p. 44) object by opening the file and processing the examples in there.

Parameters:

fileName is the name of the KEEL/WEKA formatted input file.

readAttInfo indicates whether the information of the attributes has to be read.

repUnknownVal indicates whether the unknown values need to be replaced.

realAttributesFunction indicates the type of distance function for continuous attributes.

nominalAttributesFunction indicates the type of distance function for nominal attributes.

5.8.2.2 ExtendedDataset::~ExtendedDataset ()

Destructs the **ExtendedDataset** (p. 44) object.

5.8.3 Member Function Documentation

5.8.3.1 void ExtendedDataset::deleteExamplesPerClass ()

Deletes the structures that maintain the examples organized per class.

5.8.3.2 float* ExtendedDataset::getDatasetCharacteristics ()

Returns a float* with the characteristics of the data set: number of examples, number of attributes, number of continuous attributes, number of integer attributes, number of nominal attributes, number of classes, % missing attributes, % missing examples, % missing values, % examples of the majority class, and % examples of the minority class.

Returns:

A float* with the aforementioned information.

5.8.3.3 void ExtendedDataset::organizePerClass ()

Organizes the examples per class.

It is needed before calling the stratified cross-validation method.

5.8.3.4 void ExtendedDataset::stratifiedCrossValidation (int *k*, std::string *outName*)

Splits the data set into k stratified sets following the k-fold cross-validation approach.

Parameters:

k is the number of folds.

outName is the output file name.

5.8.3.5 void ExtendedDataset::crossValidation (int *k*, std::string *outName*)

Splits the data set into k sets following the k-fold cross-validation approach.

Parameters:

k is the number of folds.

outName is the output file name.

5.8.3.6 std::string∗ ExtendedDataset::generate2ClassDatasets (std::string *baseName*)

Generates m two-class data sets from an m-class data set.

Parameters:

baseName is the base name of the data sets that will be generated.

5.8.3.7 void ExtendedDataset::print (bool *printExamples* = `false`)

Prints the data set to the standard output in KEEL (super set of WEKA) format.

Parameters:

printExamples indicates whether the examples have to be printed.

See also:

printOneClassAgainstOthers (p. 48).

5.8.3.8 std::ofstream& ExtendedDataset::print (std::ofstream & *fout*, bool *printNormalizedData*)

Prints the data set to the file referenced by fout in KEEL format.

Parameters:

fout is the output file.

printNormalizedData indicates whether data have to be printed normalized.

Returns:

An ofstream as a reference to the file.

5.8.3.9 std::ofstream& ExtendedDataset::printOneClassAgainstOthers (std::ofstream & *fout*, int *whichClass*)

Prints a two-class data set where all the classes except for whichClass are grouped in a class label groupClass.

Parameters:

fout is the output file.

whichClass is the class that will be printed against the groupClass.

Returns:

An ofstream as a reference to the file.

See also:

generate2ClassDatasets (p. 47).

5.8.3.10 void ExtendedDataset::instantiateDistanceFunctions (int *realAttributesFunction*, int *nominalAttributesFunction*)

Instantiates the distance functions.

Parameters:

realAttributesFunction is the type of distance function for continuous attributes.

nominalAttributesFunction is the type of distance function for nominal attriubtes.

5.8.3.11 void ExtendedDataset::printExample (std::ofstream & *fout*, float ∗ *inst*, int *action*) `[protected]`

Prints a single example to the output file.

Parameters:

fout is the output file where the example will be printed.

inst is the example to be printed.

action is the class of the example.

5.8.3.12 void ExtendedDataset::printExampleNoNormalized (std::ofstream & *fout*, float ∗ *inst*, int *action*) `[protected]`

Prints a single example (no normalized) to the output file.

Parameters:

fout is the output file where the example will be printed.

inst is the example to be printed.

action is the class of the example.

5.8.3.13 void ExtendedDataset::printExample (float ∗ *inst*, int *action*) `[protected]`

Prints a single example to the standard output.

Parameters:

inst is the example to be printed.
action is the class of the example.

5.8.3.14 void ExtendedDataset::printHeader (std::ofstream & *fout*) `[protected]`

Prints the header of the data set.

Parameters:

fout is the output file where the header will be printed.

5.8.3.15 int ExtendedDataset::INDEX (int *i*, int *j*) `[protected]`

Calculates the index in the distance array.

Parameters:

i is the index of the first example.
j is the index of the second example.

Returns:

An integer with the distance between i and j.

5.8.3.16 float ExtendedDataset::getDistance (int *i*, int *j*) `[protected]`

Returns the distance between two examples.

Parameters:

i is the position of the first example.
j is the position of the second example.

Returns:

A float with the distance between i and j.

5.8.3.17 float ExtendedDataset::getDistance (float ∗ *ex1*, float ∗ *ex2*) `[protected]`

Returns the distance between two examples.

Parameters:

ex1 is the first example.
ex2 is the second example.

Returns:

A float with the approximate distance between ex1 and ex2.

5.8.3.18 float ExtendedDataset::getApproximateDistance (int *i*, int *j*) [protected]

Returns the distance but without applying the last square root.

Thus, this is not the exact distance but enables to speed up the code.

Parameters:

i is the position of the first example.
j is the position of the second example.

Returns:

A float with the approximate distance between i and j.

5.8.3.19 float ExtendedDataset::getApproximateDistance (float ∗ *ex1*, float ∗ *ex2*) [protected]

Returns the distance but without applying the last square root.

Thus, this is not the exact distance but enables to speed up the code.

Parameters:

ex1 is the first example.
ex2 is the second example.

Returns:

A float with the approximate distance between ex1 and ex2.

5.8.4 Field Documentation

5.8.4.1 int ExtendedDataset::typeOfContinuousDistance [protected]

Type of distance function used for continuous attributes.

5.8.4.2 int ExtendedDataset::typeOfNominalDistance [protected]

Type of distance function used for nominal attributes.

5.8.4.3 float ExtendedDataset::MAXIMUM_EXAMPLES_FOR_-PRECALCULATION [static]

Maximum examples for precalculation.

5.8.4.4 int ExtendedDataset::EUCLIDEAN [static]

Option to select the Euclidean distance function for continuous attributes.

5.8.4.5 int ExtendedDataset::NORMALIZED_EUCLIDEAN [static]

Option to select the normalized Euclidean distance function for continuous attributes.

5.8.4.6 int ExtendedDataset::STD_WEIGHTED_EUCLIDEAN `[static]`

Option to select the Euclidean distance function weighted by standard deviation for continuous attributes.

5.8.4.7 int ExtendedDataset::OVERLAP_NOMINAL `[static]`

Option to select the overlap distance function for nominal attributes.

5.8.4.8 int ExtendedDataset::VDM_NOMINAL `[static]`

Option to select the VDM distance function for nominal attributes.

5.8.4.9 int* ExtendedDataset::numberOfExamplesPerClass `[protected]`

Number of examples per each class.

5.8.4.10 float*** ExtendedDataset::examplesPerClass `[protected]`

Examples organized per class (index examplesPerClass[class][example]).

5.8.4.11 int** ExtendedDataset::indexExamplesPerClass `[protected]`

Corresponding position of the example in the example vector (index indexExamplesPerClass[class][example]).

5.8.4.12 DistanceFunction** ExtendedDataset::distanceFunction `[protected]`

Distance functions for each attribute.

5.8.4.13 float* ExtendedDataset::distances `[protected]`

Distances between each pair of examples.

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

- ExtendedDataset.h

5.9 HeapTree< Elem > Class Template Reference

Implements a max-heap.

`#include <Heap.h>`

Public Member Functions

- **HeapTree** (int mSize)
- **HeapTree** (const **HeapTree**< Elem > &otherTree)
- ∼**HeapTree** ()
- bool **add** (const Elem &item)
- Elem **remove** ()
- void **removeAll** ()
- int **getNumberOfElements** ()
- Elem & **getElementAt** (int i)
- Elem & **getFirst** ()
- void **print** ()

5.9.1 Detailed Description

template<class Elem> class HeapTree< Elem >

Implements a max-heap.

This class implements a max-heap and is used by the kNN algorithm.

Author:

Albert Orriols-Puig and Nuria Macia
Grup de Recerca en Sistemes Intel.ligents
La Salle - Universitat Ramon Llull
C/ Quatre Camins, 2. 08022, Barcelona (Spain)

Date:

Created April, 2009
Last modified December, 2010

5.9.2 Constructor & Destructor Documentation

5.9.2.1 template<class Elem> HeapTree< Elem >::HeapTree (int *mSize*) `[inline]`

Constructs a new **HeapTree** (p. 52) object of maxSize elements (100 by default).

Parameters:

mSize is the size of the tree.

5.9.2.2 template<class Elem> HeapTree< Elem >::HeapTree (const HeapTree< Elem > & *otherTree*) `[inline]`

Copy constructor.

Parameters:

otherTree is the tree to be copied to the new object.

References HeapTree< Elem >::data, HeapTree< Elem >::maxSize, and HeapTree< Elem >::numberOfElements.

5.9.2.3 template<class Elem> HeapTree< Elem >::~HeapTree () `[inline]`

Destructs the **HeapTree** (p. 52) object.

5.9.3 Member Function Documentation

5.9.3.1 template<class Elem> bool HeapTree< Elem >::add (const Elem & *item*) `[inline]`

Adds one element to the heap.

Parameters:

item is the element to be inserted.

Returns:

A Boolean that indicates whether the insertion is successful.

Return values:

true the insertion was successful.

false the element was not inserted.

5.9.3.2 template<class Elem> Elem HeapTree< Elem >::remove () `[inline]`

Returns and removes the first element of the heap.

Returns:

An element which is the biggest element in the heap.

5.9.3.3 template<class Elem> void HeapTree< Elem >::removeAll () `[inline]`

Removes all the elements of the heap.

5.9.3.4 template<class Elem> int HeapTree< Elem >::getNumberOfElements () `[inline]`

Returns the number of elements in the heap.

Returns:

An integer with the number of elements in the heap.

5.9.3.5 template<class Elem> Elem & HeapTree< Elem >::getElementAt (int *i*) `[inline]`

Returns the element at the position specified by the input parameter.

Parameters:

i is the position to be consulted.

Returns:

The element placed in the position i.

References Utils::logWarningError().

5.9.3.6 template<class Elem> Elem & HeapTree< Elem >::getFirst () `[inline]`

Returns the element at the first position.

Returns:

The element placed in the first position of the heap.

References Utils::logWarningError().

5.9.3.7 template<class Elem> void HeapTree< Elem >::print () `[inline]`

Prints the heap.

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

- Heap.h

5.10 InputOptions Class Reference

Parses and stores the command line options.

`#include <InputOptions.h>`

Public Member Functions

- **InputOptions** ()
- ~**InputOptions** ()
- void **parseInput** (int argc, char **argv)
- bool **isAnyOptionSelected** ()
- bool **isAnyComplexityMeasureSelected** ()
- bool **isIncompatibleOptions** ()
- void **printCommandLineExample** (const char *invalidOption)
- void **printUsage** ()
- std::string **getInputDatasetName** ()
- std::string **getOutputDatasetName** ()
- bool **getLatexOutput** ()
- bool **getXMLOutput** ()
- bool **getTransformTo2ClassData** ()
- bool **getRunCrossValidation** ()
- int **getFoldsCV** ()
- bool **getRunInMultipleDatasetMode** ()
- bool **getRunAllComplexityMeasures** ()
- bool **getF1** ()
- bool **getF1v** ()
- bool **getF2** ()
- bool **getF3** ()
- bool **getF4** ()
- bool **getL1** ()
- bool **getL2** ()
- bool **getL3** ()
- bool **getN1** ()
- bool **getN2** ()
- bool **getN3** ()
- bool **getN4** ()
- bool **getT1** ()
- bool **getT2** ()
- int **getNumberOfComplexityMeasuresToCompute** ()
- bool **doDiscriminateClasses** ()
- bool **getPrintNormalizedDataset** ()
- int **getTypeOfContinuousDistFunction** ()
- int **getTypeOfNominalDistFunction** ()
- bool **getReplaceUnknownValues** ()
- bool **getShowGPLInfoNoWarr** ()
- bool **getShowGPLInfoRedistribution** ()
- void **setInputDatasetName** (std::string datasetName)
- void **setOutputDatasetName** (std::string datasetName)

5.10.1 Detailed Description

Parses and stores the command line options.

This class parses and maintains the command line parameters for the configuration.

Author:

Albert Orriols-Puig and Nuria Macia
Grup de Recerca en Sistemes Intel.ligents
La Salle - Universitat Ramon Llull
C/ Quatre Camins, 2. 08022, Barcelona (Spain)

Date:

Created April, 2009
Last modified December, 2010

5.10.2 Constructor & Destructor Documentation

5.10.2.1 InputOptions::InputOptions ()

Constructs a new **InputOptions** (p. 55) object.

5.10.2.2 InputOptions::∼InputOptions ()

Destructs the **InputOptions** (p. 55) object.

5.10.3 Member Function Documentation

5.10.3.1 void InputOptions::parseInput (int *argc*, char ∗∗ *argv*)

Parses the input options specified from the command line.

Parameters:

argc is the argument counter.

argv contains the command line arguments.

5.10.3.2 bool InputOptions::isAnyOptionSelected ()

Indicates whether at least one option has been selected.

Returns:

A Boolean that indicates whether any option has been selected.

Return values:

true at least one option has been selected.

false no options have been selected.

5.10.3.3 bool InputOptions::isAnyComplexityMeasureSelected ()

Checks whether any complexity measure has been selected.

Returns:

A Boolean that indicates whether any complexity measure is selected.

Return values:

true at least one complexity measure has been selected.

false no complexity measures have been selected.

5.10.3.4 bool InputOptions::isIncompatibleOptions ()

Checks whether there are incompatible options selected.

In case of incompatibility among options, it generates an error message to duly inform the user.

Returns:

A Boolean that indicates whether any option is incompatible.

Return values:

true selected options are incompatible.

false selected options are well-defined.

5.10.3.5 void InputOptions::printCommandLineExample (const char * *invalidOption*)

Shows a list of all the possible configuration parameters.

Parameters:

invalidOption is a string with the invalid option(s) introduced by the user.

5.10.3.6 void InputOptions::printUsage ()

Prints the application usage mode.

5.10.3.7 std::string InputOptions::getInputDatasetName () `[inline]`

Returns the name of the input data set.

Returns:

A string with the name of the input data set.

5.10.3.8 std::string InputOptions::getOutputDatasetName () `[inline]`

Returns the name of the output data set.

Returns:

A string with the name of the output data set.

5.10.3.9 bool InputOptions::getLatexOutput () `[inline]`

Returns whether the user has selected the option of saving the results in a latex file.

Returns:

A Boolean that indicates whether the user has selected the option of saving the results in a latex file.

Return values:

true latex output file has been selected.

false latex output file has not been selected.

5.10.3.10 bool InputOptions::getXMLOutput () `[inline]`

Returns whether the user has selected the option of saving the results in a XML file.

Returns:

A Boolean that indicates whether the user has selected the option of saving the results in a XML file.

Return values:

true XML output file has been selected.

false XML output file has not been selected.

5.10.3.11 bool InputOptions::getTransformTo2ClassData () `[inline]`

Returns whether the user has selected the option of transforming an m-class data set into m different two-class data sets.

Returns:

A Boolean that indicates whether the user has selected the option of transforming the data set into different two-class data sets.

Return values:

true transforming option has been selected.

false transforming option has not been selected.

5.10.3.12 bool InputOptions::getRunCrossValidation () `[inline]`

Returns whether the user has selected the option of running cross-validation.

Returns:

A Boolean that indicates whether the user has selected the option of running cross-validation.

Return values:

true cross-validation option has been selected.

false cross-validation option has not been selected.

5.10.3.13 int InputOptions::getFoldsCV () `[inline]`

Returns the number of folds in cross-validation.

Returns:

An integer with the number of folds in cross-validation.

5.10.3.14 bool InputOptions::getRunInMultipleDatasetMode () `[inline]`

Returns whether the user has selected the option of running the measures in batch mode.

Returns:

A Boolean that indicates whether the user has selected the option of running the measures in batch mode.

Return values:

true batch mode has been selected.

false batch mode has not been selected.

5.10.3.15 bool InputOptions::getRunAllComplexityMeasures () `[inline]`

Returns whether the user has selected the option of running all the complexity measures.

Returns:

A Boolean that indicates whether the user has selected the option of running all the complexity measures.

Return values:

true option of running all the complexity measures has been selected.

false specific complexity measures have been selected to be run.

5.10.3.16 bool InputOptions::getF1 () `[inline]`

Returns whether the user has selected to run the maximum Fisher's discriminant ratio (measure F1).

Returns:

A Boolean that indicates whether the user has selected to run F1.

Return values:

true measure F1 has been selected to be run.

false measure F1 has not been selected to be run.

5.10.3.17 bool InputOptions::getF1v () `[inline]`

Returns whether the user has selected to run the vector-directed version of the maximum Fisher's discriminant ratio (measure F1v).

Returns:

A Boolean that indicates whether the user has selected to run F1v.

Return values:

true measure F1v has been selected to be run.

false measure F1v has not been selected to be run.

5.10.3.18 bool InputOptions::getF2 () `[inline]`

Returns whether the user has selected to run the overlap of per-class bounding boxes (measure F2).

Returns:

A Boolean that indicates whether the user has selected to run F2.

Return values:

true measure F2 has been selected to be run.

false measure F2 has not been selected to be run.

5.10.3.19 bool InputOptions::getF3 () `[inline]`

Returns whether the user has selected to run the maximum (individual) feature efficiency (measure F3).

Returns:

A Boolean that indicates whether the user has selected to run F3.

Return values:

true measure F3 has been selected to be run.

false measure F3 has not been selected to be run.

5.10.3.20 bool InputOptions::getF4 () `[inline]`

Returns whether the user has selected to run the collective feature efficiency (measure F4).

Returns:

A Boolean that indicates whether the user has selected to run F4.

Return values:

true measure F4 has been selected to be run.

false measure F4 has not been selected to be run.

5.10.3.21 bool InputOptions::getL1 () `[inline]`

Returns whether the user has selected to run the minimized sum of the error distance of a linear classifier (measure L1).

Returns:

A Boolean that indicates whether the user has selected to run L1.

Return values:

true measure L1 has been selected to be run.

false measure L1 has not been selected to be run.

5.10.3.22 bool InputOptions::getL2 () `[inline]`

Returns whether the user has selected to run the training error of a linear classifier (measure L2).

Returns:

A Boolean that indicates whether the user has selected to run L2.

Return values:

true measure L2 has been selected to be run.

false measure L2 has not been selected to be run.

5.10.3.23 bool InputOptions::getL3 () `[inline]`

Returns whether the user has selected to run the nonlinearity of a linear classifier (measure L3).

Returns:

A Boolean that indicates whether the user has selected to run L3.

Return values:

true measure L3 has been selected to be run.

false measure L3 has not been selected to be run.

5.10.3.24 bool InputOptions::getN1 () `[inline]`

Returns whether the user has selected to run the fraction of points on the class boundary (measure N1).

Returns:

A Boolean that indicates whether the user has selected to run N1.

Return values:

true measure N1 has been selected to be run.

false measure N1 has not been selected to be run.

5.10.3.25 bool InputOptions::getN2 () `[inline]`

Returns whether the user has selected to run the ratio of average intra/inter class nearest neighbor distance (measure N2).

Returns:

A Boolean that indicates whether the user has selected to run N2.

Return values:

true measure N2 has been selected to be run.

false measure N2 has not been selected to be run.

5.10.3.26 bool InputOptions::getN3 () `[inline]`

Returns whether the user has selected to run the leave-one-out error rate of the one-nearest neighbor classifier (measure N3).

Returns:

A Boolean that indicates whether the user has selected to run N3.

Return values:

true measure N3 has been selected to be run.

false measure N3 has not been selected to be run.

5.10.3.27 bool InputOptions::getN4 () `[inline]`

Returns whether the user has selected to run the nonlinearity of the one-nearest neighbor classifier (measure N4).

Returns:

A Boolean that indicates whether the user has selected to run N4.

Return values:

true measure N4 has been selected to be run.

false measure N4 has not been selected to be run.

5.10.3.28 bool InputOptions::getT1 () `[inline]`

Returns whether the user has selected to run the fraction of maximum covering spheres (measure T1).

Returns:

A Boolean that indicates whether the user has selected to run T1.

Return values:

true measure T1 has been selected to be run.

false measure T1 has not been selected to be run.

5.10.3.29 bool InputOptions::getT2 () `[inline]`

Returns whether the user has selected to run the average number of points per dimension (measure T2).

Returns:

A Boolean that indicates whether the user has selected to run T2.

Return values:

true measure T2 has been selected to be run.

false measure T2 has not been selected to be run.

5.10.3.30 int InputOptions::getNumberOfComplexityMeasuresToCompute () `[inline]`

Returns the number of complexity measures to compute.

Returns:

An interger with the number of complexity measures to compute.

5.10.3.31 bool InputOptions::doDiscriminateClasses () `[inline]`

Returns whether the classes need to be discriminated when applying the complexity measures.

Returns:

A Boolean that indicates whether the classes have to be discriminated for multiple class data sets.

Return values:

true classes have to be discriminated for multiple class data sets.
false classes have not to be discriminated for multiple class data sets.

5.10.3.32 bool InputOptions::getPrintNormalizedDataset () `[inline]`

Returns whether the data set needs to be printed.

Returns:

A Boolean that indicates whether the user has selected the option of printing the data set.

Return values:

true the option of printing the data set has been selected.
false the option of printing the data set has not been selected.

5.10.3.33 int InputOptions::getTypeOfContinuousDistFunction () `[inline]`

Returns the type of distance function for continuous attributes.

Returns:

An integer with the type of distance function for continuous attributes.

Return values:

1 Euclidean distance function.
2 normalized Euclidean distance function (default option).
3 Euclidean distance function weighted by the standard deviation.

5.10.3.34 int InputOptions::getTypeOfNominalDistFunction () `[inline]`

Returns the type of distance function for nominal attributes.

Returns:

An integer with the type of distance function for nominal attributes.

Return values:

1 overlap distance function (default option).
2 VDM distance function.
3 Euclidean distance function.
4 normalized Euclidean distance function.

5.10.3.35 bool InputOptions::getReplaceUnknownValues () `[inline]`

Returns whether the unknown values have to be replaced.

Returns:

A Boolean that indicates whether the unknown values have to be replaced.

Return values:

true unknown values have to be replaced.

false unknown values do not have to be replaced.

5.10.3.36 bool InputOptions::getShowGPLInfoNoWarr () `[inline]`

Returns whether the GPL information about ABSOLUTELY NO WARRANTY has to be shown.

Returns:

A Boolean that indicates whether the GPL information about ABSOLUTELY NO WARRANTY has to be shown.

Return values:

true GPL information about ABSOLUTELY NO WARRANTY has to be shown.

false GPL information about ABSOLUTELY NO WARRANTY must not be shown.

5.10.3.37 bool InputOptions::getShowGPLInfoRedistribution () `[inline]`

Returns whether the GPL information about redistribution has to be shown.

Returns:

A Boolean that indicates whether the GPL information about redistribution has to be shown.

Return values:

true GPL information about redistribution has to be shown.

false GPL information about redistribution must not be shown.

5.10.3.38 void InputOptions::setInputDatasetName (std::string *datasetName*) `[inline]`

Sets the name of the input data set.

Parameters:

datasetName is the name of the input data set.

5.10.3.39 void InputOptions::setOutputDatasetName (std::string *datasetName*) `[inline]`

Sets the name of the output data set.

Parameters:

datasetName is the name of the output data set.

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

- InputOptions.h

5.11 Matrix Class Reference

Implements methods to store, transform, and operate with matrices.

`#include <Matrix.h>`

Public Member Functions

- **Matrix** ()
- **Matrix** (int nr, int nc)
- **Matrix** (const **Matrix** &m)
- ∼**Matrix** ()
- float & **getReference** (int i, int j)
- float **getValue** (int i, int j)
- void **setValue** (int i, int j, float value)
- **Matrix computeInverse** ()
- void **computeGaussJordan** ()
- void **swapRows** (int ∗index, int i, int j)
- void **resortMatrix** (int ∗index)
- **Matrix transpose** ()
- **Matrix operator-** (const **Matrix** &m)
- **Matrix operator+** (const **Matrix** &m)
- **Matrix operator∗** (const **Matrix** &m)
- **Matrix multScalar** (float scalar)
- **Matrix computePseudoInverse** ()
- void **transformDiagonalMatrixToInverse** ()
- void **computeSVD** (**Matrix** ∗&matW, **Matrix** ∗&matV)
- void **decomposeSVD** (float ∗&W, float ∗∗&V)
- void **reorder** (float ∗&W, float ∗∗&V)
- float **pythag** (float par1, float par2)
- float **SQR** (float x)
- float **SIGN** (float par1, float par2)
- float **FMAX** (float par1, float par2)
- int **IMIN** (int par1, int par2)

Friends

- std::ostream & **operator<<** (std::ostream &os, **Matrix** &m)

5.11.1 Detailed Description

Implements methods to store, transform, and operate with matrices.

This class enables the user to create matrices and apply certain operations.

Author:

Albert Orriols-Puig and Nuria Macia
Grup de Recerca en Sistemes Intel.ligents
La Salle - Universitat Ramon Llull
C/ Quatre Camins, 2. 08022, Barcelona (Spain)

Date:

Created April, 2009
Last modified December, 2010

5.11.2 Constructor & Destructor Documentation

5.11.2.1 Matrix::Matrix () `[inline]`

Constructs a new **Matrix** (p. 67) object.

5.11.2.2 Matrix::Matrix (int *nr*, int *nc*)

Constructs a new **Matrix** (p. 67) object.

Parameters:

nr is the number of rows.
nc is the number of columns.

5.11.2.3 Matrix::Matrix (const Matrix & *m*)

Copy constructor.

Parameters:

m is the matrix to be copied.

5.11.2.4 Matrix::∼Matrix ()

Destructs the **Matrix** (p. 67) object.

5.11.3 Member Function Documentation

5.11.3.1 float& Matrix::getReference (int *i*, int *j*)

Returns a reference to the position indicated by the indeces.

Parameters:

i is the row index.
j is the column index.

Returns:

A float reference of the position indicated by the indeces.

5.11.3.2 float Matrix::getValue (int *i*, int *j*)

Returns the value of the position indicated by the indeces.

Parameters:

i is the row index.
j is the column index.

Returns:

A float with the position indicated by the indeces.

5.11.3.3 void Matrix::setValue (int *i*, int *j*, float *value*)

Sets the value to the specified position of the matrix.

Parameters:

i is the row index.
j is the column index.
value is the value that has to be set.

5.11.3.4 Matrix Matrix::computeInverse ()

Computes the inverse of the matrix with the Gauss Jordan procedure.

Returns:

A **Matrix** (p. 67) with the inverse of the original matrix.

5.11.3.5 void Matrix::computeGaussJordan ()

Computes the Gauss Jordan elimination on the matrix.

5.11.3.6 void Matrix::swapRows (int ∗ *index*, int *i*, int *j*)

Swaps the two rows passed as argument.

Parameters:

index is the vector where the rows will be swapped.
i is the first row.
j is the second row.

5.11.3.7 void Matrix::resortMatrix (int * *index*)

Resorts the matrix rows according to the indeces passed as argument.

Parameters:

index contains the indeces to resort.

5.11.3.8 Matrix Matrix::transpose ()

Returns the transposed matrix.

Returns:

A **Matrix** (p. 67) with the transposed matrix.

5.11.3.9 Matrix Matrix::operator- (const Matrix & *m*)

Computes the difference of matrices.

Parameters:

m is the second matrix of the difference.

Returns:

A **Matrix** (p. 67) with the new matrix.

5.11.3.10 Matrix Matrix::operator+ (const Matrix & *m*)

Computes the sum of matrices.

Parameters:

m is the second matrix of the sum.

Returns:

A **Matrix** (p. 67) with the new matrix.

5.11.3.11 Matrix Matrix::operator* (const Matrix & *m*)

Computes the product of matrices.

Parameters:

m is the second matrix of the product.

Returns:

A **Matrix** (p. 67) with the new matrix.

5.11.3.12 Matrix Matrix::multScalar (float *scalar*)

Multiplies the matrix by a scalar number.

Parameters:

scalar is the scalar that multiplies the matrix.

Returns:

A **Matrix** (p. 67) having multiplied the original one with the scalar number.

5.11.3.13 Matrix Matrix::computePseudoInverse ()

Computes the pseudo inverse of the matrix.

The object of the class is modified.

Returns:

A **Matrix** (p. 67) with the pseudo inverse matrix.

See also:

computeSVD (p. 71).

5.11.3.14 void Matrix::transformDiagonalMatrixToInverse ()

Transforms a diagonal square matrix into its inverse by replacing each element of the diagonal with its reciprocal.

The matrix is overwritten.

5.11.3.15 void Matrix::computeSVD (Matrix *& *matW*, Matrix *& *matV*)

Computes the singular value decomposition matrix = U·W·V' where (U is mxn, W is nxn, and V is nxn).

The matrix U replaces matrix (the class attribute) on output.

Parameters:

matW is the diagonal matrix W of the decomposition.

matV is the matrix V of size nxn. Note that it is not the transpose matrix.

See also:

calculateSVD.

5.11.3.16 void Matrix::decomposeSVD (float *& *W*, float **& *V*)

Computes the singular value decomposition matrix = U*W*V' where (U is mxn, W is nxn, and V is nxn).

The matrix U replaces matrix (the class attribute) on output. The diagnoal matrix W is output as a vector of size n. The matrix V (not the transpose) is computed.

Parameters:

W is a vector of size n that represents the diagonal of matrix W [output parameter].

V is the matrix V of size nxn. Note that it is not the transpose matrix.

5.11.3.17 void Matrix::reorder (float ∗& *W*, float ∗∗& *V*)

The routine sorts the singular values, and the corresponding values of U and V, by decreasing magnitude.

In addition, in order to maximize the number of positive elements, signs of corresponding columns are flipped.

5.11.3.18 float Matrix::pythag (float *par1*, float *par2*)

Computes (par1^2 + par2^2)^0.5 without destructive underflow or overflow.

Parameters:

par1 is the first parameter.

par2 is the second parameter.

Returns:

A float with (par1^2 + par2^2)^0.5 without destructive underflow or overflow.

5.11.3.19 float Matrix::SQR (float *x*) `[inline]`

Computes the square of a float.

Parameters:

x is the argument.

Returns:

A float with x^2.

5.11.3.20 float Matrix::SIGN (float *par1*, float *par2*) `[inline]`

Returns the first parameter as positive if the second one is positive.

Otherwise, returns the negative value of the second parameter.

Parameters:

par1 is the first parameter.

par2 is the second parameter.

Returns:

A float with the positive or negative value of the first parameter according to the sign of the second parameter.

5.11.3.21 float Matrix::FMAX (float *par1*, float *par2*) `[inline]`

Returns the maximum of the two parameters.

Parameters:

par1 is the first parameter.

par2 is the second parameter.

Returns:

A float with the maximum value between par1 and par2.

5.11.3.22 int Matrix::IMIN (int *par1*, int *par2*) `[inline]`

Returns the minimum of the two parameters.

Parameters:

par1 is the first parameter.

par2 is the second parameter.

Returns:

An integer with the maximum value between par1 and par2.

5.11.4 Friends And Related Function Documentation

5.11.4.1 std::ostream& operator<< (std::ostream & *os*, Matrix & *m*) `[friend]`

Overloads the standard output.

Parameters:

os is a reference to the standard output.

m is a reference to the matrix that has to be printed.

Returns:

A reference to ostream.

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

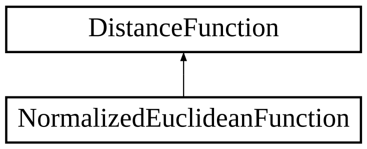
- Matrix.h

5.12 NormalizedEuclideanFunction Class Reference

Implements the normalized Euclidean distance function for continuous attributes.

`#include <NormalizedEuclideanFunction.h>`

Inheritance diagram for NormalizedEuclideanFunction::



Public Member Functions

- **NormalizedEuclideanFunction** (float min, float max)
- ~**NormalizedEuclideanFunction** ()
- float **computeDistance** (float att1, float att2)

5.12.1 Detailed Description

Implements the normalized Euclidean distance function for continuous attributes.

This is an implementation of the **DistanceFunction** (p. 37) abstract class for continuous attributes. It implements the normalized Euclidean distance.

Author:

Albert Orriols-Puig and Nuria Macia
Grup de Recerca en Sistemes Intel.ligents
La Salle - Universitat Ramon Llull
C/ Quatre Camins, 2. 08022, Barcelona (Spain)

Date:

Created April, 2009
Last modified December, 2010

5.12.2 Constructor & Destructor Documentation

5.12.2.1 NormalizedEuclideanFunction::NormalizedEuclideanFunction (float *min*, float *max*)

Constructs a new **NormalizedEuclideanFunction** (p. 74) object.

Parameters:

min is the minimum value that the attribute can take.

max is the maximum value that the attribute can take.

5.12.2.2 NormalizedEuclideanFunction::~NormalizedEuclideanFunction ()

Destructs the **NormalizedEuclideanFunction** (p. 74) object.

5.12.3 Member Function Documentation

5.12.3.1 float NormalizedEuclideanFunction::computeDistance (float *att1*, float *att2*) `[virtual]`

Computes the distance between two attribute values.

Parameters:

att1 is the value of attribute 1.

att2 is the value of attribute 2.

Returns:

A float with the distance between att1 and att2.

Implements **DistanceFunction** (p. 38).

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

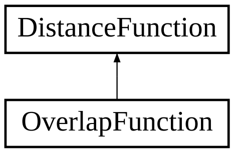
- DistanceFunctions/NormalizedEuclideanFunction.h

5.13 OverlapFunction Class Reference

Implements the overlap distance function for nominal attributes.

`#include <OverlapFunction.h>`

Inheritance diagram for OverlapFunction::



Public Member Functions

- **OverlapFunction** ()
- ~**OverlapFunction** ()
- float **computeDistance** (float att1, float att2)

5.13.1 Detailed Description

Implements the overlap distance function for nominal attributes.

This is an implementation of the **DistanceFunction** (p. 37) abstract class for nominal attributes. It implements the overlap distance. This distance sets 0 to the distance of two attributes with the same values and 1 otherwise.

Author:

Albert Orriols-Puig and Nuria Macia
Grup de Recerca en Sistemes Intel.ligents
La Salle - Universitat Ramon Llull
C/ Quatre Camins, 2. 08022, Barcelona (Spain)

Date:

Created April, 2009
Last modified December, 2010

5.13.2 Constructor & Destructor Documentation

5.13.2.1 OverlapFunction::OverlapFunction ()

Constructs a **OverlapFunction** (p. 76) object.

5.13.2.2 OverlapFunction::~OverlapFunction ()

Destructs the **OverlapFunction** (p. 76) object.

5.13.3 Member Function Documentation

5.13.3.1 float OverlapFunction::computeDistance (float *att1*, float *att2*) `[virtual]`

Computes the distance between two attribute values.

Parameters:

att1 is the value of attribute 1.

att2 is the value of attribute 2.

Returns:

A float with the distance between att1 and att2.

Implements **DistanceFunction** (p. 38).

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

- DistanceFunctions/OverlapFunction.h

5.14 ResultsContainer Class Reference

Stores the results.

`#include <ResultsContainer.h>`

Public Member Functions

- **ResultsContainer** ()
- ∼**ResultsContainer** ()
- void **addElement** (std::string datasetName, float ∗elem)
- float ∗ **getResult** (int position)
- std::string **getDatasetName** (int position)
- std::string **getShortDatasetName** (int position, int maxSize)
- int **getNumberOfDatasets** ()

5.14.1 Detailed Description

Stores the results.

This class stores the results of runs over a single or multiple data sets. It maintains the names of the data sets and the complexity measures results for each one.

Author:

Albert Orriols-Puig and Nuria Macia
Grup de Recerca en Sistemes Intel.ligents
La Salle - Universitat Ramon Llull
C/ Quatre Camins, 2. 08022, Barcelona (Spain)

Date:

Created April, 2009
Last modified December, 2010

5.14.2 Constructor & Destructor Documentation

5.14.2.1 ResultsContainer::ResultsContainer () `[inline]`

Constructs a new **ResultsContainer** (p. 78) object.

5.14.2.2 ResultsContainer::∼ResultsContainer () `[inline]`

Destructs the **ResultsContainer** (p. 78) object.

5.14.3 Member Function Documentation

5.14.3.1 void ResultsContainer::addElement (std::string *datasetName*, float ∗ *elem*) `[inline]`

Adds a specific element to the result container.

Parameters:

datasetName is the name of the data set for which the result is provided.
elem is a float∗ with the results.

5.14.3.2 float∗ ResultsContainer::getResult (int *position*) `[inline]`

Returns the result of the consulted data sets.

Parameters:

position is the position of the data set to be consulted.

Returns:

A float∗ with the results.

5.14.3.3 std::string ResultsContainer::getDatasetName (int *position*) `[inline]`

Returns the data set name of a specific data set.

Parameters:

position is the position of the data set to be consulted.

Returns:

A string with the data set name.

5.14.3.4 std::string ResultsContainer::getShortDatasetName (int *position*, int *maxSize*) `[inline]`

Returns a short of the data set name of a specific data set.

Parameters:

position is the position of the data set to be consulted.
maxSize is the maximum size for the name.

Returns:

A string with the short name of the data set.

5.14.3.5 int ResultsContainer::getNumberOfDatasets () `[inline]`

Returns the number of data sets.

Returns:

An integer with the number of data sets.

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

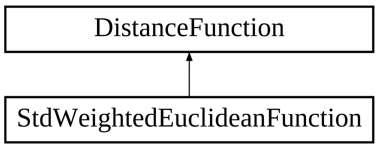
- ResultsContainer.h

5.15 StdWeightedEuclideanFunction Class Reference

Implements the standard-deviation weighted Euclidean distance for continuous attributes.

`#include <StdWeightedEuclideanFunction.h>`

Inheritance diagram for StdWeightedEuclideanFunction::



Public Member Functions

- **StdWeightedEuclideanFunction** (double std)
- ∼**StdWeightedEuclideanFunction** ()
- float **computeDistance** (float att1, float att2)

5.15.1 Detailed Description

Implements the standard-deviation weighted Euclidean distance for continuous attributes.

This is an implementation of the **DistanceFunction** (p. 37) abstract class for continuous attributes. It implements the standard-deviation weighted Euclidean distance. That is, the distance is weighted by four times the standard deviation of the attribute to reduce the effect of outliers.

Author:

Albert Orriols-Puig and Nuria Macia
Grup de Recerca en Sistemes Intel.ligents
La Salle - Universitat Ramon Llull
C/ Quatre Camins, 2. 08022, Barcelona (Spain)

Date:

Created April, 2009
Last modified December, 2010

5.15.2 Constructor & Destructor Documentation

5.15.2.1 StdWeightedEuclideanFunction::StdWeightedEuclideanFunction (double *std*)

Constructs a new **StdWeightedEuclideanFunction** (p. 81) object.

Parameters:

std is the standard deviation of the attribute.

5.15.2.2 StdWeightedEuclideanFunction::~StdWeightedEuclideanFunction ()

Destructs the **StdWeightedEuclideanFunction** (p. 81) object.

5.15.3 Member Function Documentation

5.15.3.1 float StdWeightedEuclideanFunction::computeDistance (float *att1*, float *att2*) `[virtual]`

Computes the distance between two attribute values.

Parameters:

att1 is the value of attribute 1.

att2 is the value of attribute 2.

Returns:

A float with the distance between att1 and att2.

Implements **DistanceFunction** (p. 38).

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

- DistanceFunctions/StdWeightedEuclideanFunction.h

5.16 StringTokenizer Class Reference

Implements a string tokenizer class to break strings into tokens.

`#include <StringTokenizer.h>`

Public Member Functions

- **StringTokenizer** (std::string str, std::string separator)
- **StringTokenizer** (std::string str, std::string separator, std::string sep2)
- ∼**StringTokenizer** ()
- std::string **getNextToken** ()
- int **countTokens** () const
- bool **hasMoreTokens** () const

5.16.1 Detailed Description

Implements a string tokenizer class to break strings into tokens.

This class is similar to the Java one and splits a string into tokens separated by the specified delimiters.

Author:

Albert Orriols-Puig and Nuria Macia
Grup de Recerca en Sistemes Intel.ligents
La Salle - Universitat Ramon Llull
C/ Quatre Camins, 2. 08022, Barcelona (Spain)

Date:

Created April, 2009
Last modified December, 2010

5.16.2 Constructor & Destructor Documentation

5.16.2.1 StringTokenizer::StringTokenizer (std::string *str*, std::string *separator*)

Constructs a new **StringTokenizer** (p. 83) object for the specified string.

The characters in the separator argument are the delimiters that separate tokens and will not be treated as tokens.

Parameters:

str is the string to be parsed.

separator is the delimiter character.

5.16.2.2 StringTokenizer::StringTokenizer (std::string *str*, std::string *separator*, std::string *sep2*)

Constructs a **StringTokenizer** (p. 83) object for the specified string.

The characters in the separator and sep2 argument are the delimiters that separate tokens and will not be treated as tokens.

Parameters:

str is the string to be parsed.

separator is the first type of delimiter.

sep2 is the second type of delimiter.

5.16.2.3 StringTokenizer::∼StringTokenizer ()

Destructs the **StringTokenizer** (p. 83) object.

5.16.3 Member Function Documentation

5.16.3.1 std::string StringTokenizer::getNextToken ()

Returns the next token.

Returns:

A string with the next token.

5.16.3.2 int StringTokenizer::countTokens () const

Counts the number of tokens that a string contains.

Returns:

An integer with the number of tokens contained in the parsed string.

5.16.3.3 bool StringTokenizer::hasMoreTokens () const

Indicates whether there are more tokens.

Returns:

A Boolean that indicates whether there are more tokens.

Return values:

true there are more tokens in the string.

false there are no more tokens in the string.

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

- StringTokenizer.h

5.17 Utils Class Reference

Implements utilities.

`#include <Utils.h>`

Static Public Member Functions

- static void **setSeed** (long seed)
- static void **generateNewRandomSeed** ()
- static float **f_rand** ()
- static int **i_rand** (int lowV, int upV)
- static float **f_round** (float lowV, int upV)
- static bool **readLine** (std::ifstream &fin, std::string &line)
- static bool **isAComment** (std::string &line)
- static std::string **trim** (std::string &line)
- static std::string **removeFinalSpaces** (std::string &line)
- static void **printParamsToFile** (std::string fileName)
- static void **quickSort** (float *vector, int *order, int i, int j)
- static int **partition** (float *vector, int *order, int inf, int sup)
- static void **quickSort** (**Date** *vector, int *order, int i, int j)
- static int **partition** (**Date** *vector, int *order, int inf, int sup)
- static void **printGPLInformation** (bool printWarrantyInfo, bool printRedistributionInfo)
- static int **min** (int v1, int v2)
- static int **max** (int v1, int v2)
- static float **min** (float v1, float v2)
- static float **max** (float v1, float v2)
- static bool **isNumeric** (const char *line)
- static int **roundf** (float num)
- static void **initLog** (std::string fileName)
- static void **logWarningError** (std::string logMessage)
- static void **closeLog** ()

Static Public Attributes

- static std::ofstream **fLog**
- static bool **doScreenStatistics**
- static const char **REAL_ATTRIBUTE**
- static const char **INTEGER_ATTRIBUTE**
- static const char **NOMINAL_ATTRIBUTE**

5.17.1 Detailed Description

Implements utilities.

This class implements several utilities such as sorting procedures, comparing procedures, and random seed initializers.

Author:

Albert Orriols-Puig and Nuria Macia
Grup de Recerca en Sistemes Intel.ligents
La Salle - Universitat Ramon Llull
C/ Quatre Camins, 2. 08022, Barcelona (Spain)

Date:

Created April, 2009
Last modified December, 2010

5.17.2 Member Function Documentation

5.17.2.1 static void Utils::setSeed (long *seed*) `[static]`

Sets the seed of the random number generator.

Parameters:

seed is the seed.

5.17.2.2 static void Utils::generateNewRandomSeed () `[static]`

Generates a new random seed by taking the time of the system.

5.17.2.3 static float Utils::f_rand () `[static]`

Returns a continuous random number ranging between 0 and 1.

Returns:

A float with the random number.

5.17.2.4 static int Utils::i_rand (int *lowV*, int *upV*) `[static]`

Returns an integer random number ranging between lowV and upV.

Parameters:

lowV is the lower limit of the range.
upV is the upper limit of the range.

Returns:

An integer with the random number.

5.17.2.5 static float Utils::f_round (float *lowV*, int *upV*) `[static]`

Returns a continuous random number ranging between lowV and upV.

Parameters:

lowV is the lower limit of the range.
upV is the upper limit of the range.

Returns:

A float with the random number.

5.17.2.6 static bool Utils::readLine (std::ifstream & *fin*, std::string & *line*) `[static]`

Reads one line from file.

Parameters:

fin is the input file.
line is a string to store the line that will be read (passed by reference).

Returns:

A Boolean that indicates whether it is end of file.

Return values:

true end of file.
false not end of file yet.

5.17.2.7 static bool Utils::isAComment (std::string & *line*) `[static]`

Checks whether a line is a comment.

Parameters:

line is the line to be checked.

Returns:

A Boolean that indicates whether the line is a comment.

Return values:

true the line is a comment.
false the line contains useful data.

5.17.2.8 static std::string Utils::trim (std::string & *line*) `[static]`

Trims a string.

Parameters:

line is the string to be trimmed.

Returns:

A string with the trimmed string.

5.17.2.9 static std::string Utils::removeFinalSpaces (std::string & *line*) `[static]`

Removes the blank spaces at the end of the string.

Parameters:

line is the string to be processed.

Returns:

A string without blank spaces at the end.

5.17.2.10 static void Utils::printParamsToFile (std::string *fileName*) `[static]`

Prints the configuration parameters to file.

Parameters:

fileName is the name of the file where the parameters will be printed.

5.17.2.11 static void Utils::quickSort (float * *vector*, int * *order*, int *i*, int *j*) `[static]`

Implements the quicksort procedure for arrays of floats.

Parameters:

vector is the array to be sorted.

order is an auxiliary vector that needs to be sorted with the vector one.

i is the initial index from where the vector has to be sorted.

j is the final index until which the vector has to be sorted.

5.17.2.12 static int Utils::partition (float * *vector*, int * *order*, int *inf*, int *sup*) `[static]`

Implements an auxiliary function for quickSort for arrays of floats.

Parameters:

vector is the array to be sorted.

order is an auxiliary vector that needs to be sorted with the vector one.

inf is the initial index from where the vector has to be sorted.

sup is the final index until which the vector has to be sorted.

5.17.2.13 static void Utils::quickSort (Date ∗ *vector*, int ∗ *order*, int *i*, int *j*) `[static]`

Implements the quicksort procedure for arrays of Dates.

Parameters:

vector is the array to be sorted.

order is an auxiliary vector that needs to be sorted with the vector one.

i is the initial index from where the vector has to be sorted.

j is the final index until which the vector has to be sorted.

5.17.2.14 static int Utils::partition (Date ∗ *vector*, int ∗ *order*, int *inf*, int *sup*) `[static]`

Implements an auxiliary function for quickSort for arrays of Dates.

Parameters:

vector is the array to be sorted.

order is an auxiliary vector that needs to be sorted with the vector one.

inf is the initial index from where the vector has to be sorted.

sup is the final index until which the vector has to be sorted.

5.17.2.15 static void Utils::printGPLInformation (bool *printWarrantyInfo*, bool *printRedistributionInfo*) `[static]`

Prints information about the GPL license.

Parameters:

printWarrantyInfo is a Boolean that indicates whether warranty information has to be printed.

printRedistributionInfo is a Boolean that indicates whether redistribution information has to be printed.

5.17.2.16 static int Utils::min (int *v1*, int *v2*) `[inline, static]`

Returns the minimum value between the arguments.

Parameters:

v1 is the first integer.

v2 is the second integer.

Returns:

An integer with the minimum value between v1 and v2.

5.17.2.17 static int Utils::max (int *v1*, int *v2*) `[inline, static]`

Returns the maximum value between the arguments.

Parameters:

v1 is the first integer.

v2 is the second integer.

Returns:

An integer with the maximum value between v1 and v2.

5.17.2.18 static float Utils::min (float *v1*, float *v2*) `[inline, static]`

Returns the minimum value between the arguments.

Parameters:

v1 is the first float.

v2 is the second float.

Returns:

A float with the minimum value between v1 and v2.

5.17.2.19 static float Utils::max (float *v1*, float *v2*) `[inline, static]`

Returns the maximum value between the arguments.

Parameters:

v1 is the first float.

v2 is the second float.

Returns:

A float with the maximum value between v1 and v2.

5.17.2.20 static bool Utils::isNumeric (const char * *line*) `[static]`

Returns whether the passed string contains only numbers.

Parameters:

line is the input string.

Returns:

A Boolean that indicates whether the passed string contains only numbers.

Return values:

true the passed string contains only numbers.

false the passed string contains characters and may contain numbers as well.

5.17.2.21 static int Utils::roundf (float *num*) `[inline, static]`

Rounds the float to an integer.

Parameters:

num is the float number to be rounded.

Returns:

An integer with the rounded number.

5.17.2.22 static void Utils::initLog (std::string *fileName*) `[inline, static]`

Initializes the file for warning and error logging.

Parameters:

fileName is the name of the file.

References fLog.

5.17.2.23 static void Utils::logWarningError (std::string *logMessage*) `[inline, static]`

Writes the warning/error string to the log file.

Parameters:

logMessage is the string to be written.

References fLog.

Referenced by Vector< Etype >::firstElement(), HeapTree< Elem >::getElementAt(), and HeapTree< Elem >::getFirst().

5.17.2.24 static void Utils::closeLog () `[inline, static]`

Closes the log file.

References fLog.

5.17.3 Field Documentation

5.17.3.1 std::ofstream Utils::fLog `[static]`

File for warning and error logging.

Referenced by closeLog(), initLog(), and logWarningError().

5.17.3.2 bool Utils::doScreenStatistics `[static]`

Indicates whether screen statistics must be written.

5.17.3.3 const char Utils::REAL_ATTRIBUTE `[static]`

Identifies continuous attributes.

5.17.3.4 const char Utils::INTEGER_ATTRIBUTE `[static]`

Identifies integer attributes.

5.17.3.5 const char Utils::NOMINAL_ATTRIBUTE `[static]`

Identifies nominal attributes.

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

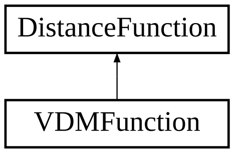
- Utils.h

5.18 VDMFunction Class Reference

Implements the VDM distance function for nominal attributes.

`#include <VDMFunction.h>`

Inheritance diagram for VDMFunction::



Public Member Functions

- **VDMFunction** (float **freqPerClass, int numOfClasses, int numOfValues)
- ~**VDMFunction** ()
- float **computeDistance** (float att1, float att2)

5.18.1 Detailed Description

Implements the VDM distance function for nominal attributes.

This is an implementation of the **DistanceFunction** (p. 37) abstract class for nominal attributes. It implements the VDM distance. This metric uses the frequency of each value per class in the training data set to determine the proximity of two values. For further details see (Randall & Martinez, 1997).

Author:

Albert Orriols-Puig and Nuria Macia
Grup de Recerca en Sistemes Intel.ligents
La Salle - Universitat Ramon Llull
C/ Quatre Camins, 2. 08022, Barcelona (Spain)

Date:

Created April, 2009
Last modified December, 2010

5.18.2 Constructor & Destructor Documentation

5.18.2.1 VDMFunction::VDMFunction (float ∗∗ *freqPerClass*, int *numOfClasses*, int *numOfValues*)

Constructs a new **VDMFunction** (p. 94) object.

Parameters:

freqPerClass is the frequency of each value of the attribute per class.

numOfClasses is the number of classes.

numOfValues is the number of values.

5.18.2.2 VDMFunction::∼VDMFunction ()

Destructs the **VDMFunction** (p. 94) object.

5.18.3 Member Function Documentation

5.18.3.1 float VDMFunction::computeDistance (float *att1*, float *att2*) `[virtual]`

Computes the distance between two attribute values.

Parameters:

att1 is the value of attribute 1.

att2 is the value of attribute 2.

Returns:

A float with the distance between att1 and att2.

Implements **DistanceFunction** (p. 38).

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

- DistanceFunctions/VDMFunction.h

5.19 Vector< Etype > Class Template Reference

Implements a **Vector** (p. 96) class like Java class **Vector** (p. 96).

`#include <Vector.h>`

Public Member Functions

- **Vector** (UINT initialCapacity=10, UINT capacityIncrement=10)
- **Vector** (const **Vector** &rhv)
- virtual ~**Vector** ()
- UINT **capacity** () const
- bool **contains** (const Etype &elem) const
- const Etype & **firstElement** () const
- int **indexOf** (const Etype &elem) const
- bool **isEmpty** () const
- const Etype & **lastElement** () const
- int **lastIndexOf** (const Etype &elem) const
- UINT **size** () const
- void **copyInto** (Etype ∗array) const
- void **addElement** (const Etype &obj)
- void **ensureCapacity** (UINT minCapacity)
- void **removeAllElements** ()
- bool **removeElement** (const Etype &obj)
- void **setSize** (UINT newSize)
- void **trimToSize** ()
- Etype & **elementAt** (UINT index) const
- void **insertElementAt** (const Etype &obj, UINT index)
- void **removeElementAt** (UINT index)
- void **removeElementAtWithoutDestroyingElement** (UINT index)
- void **setElementAt** (const Etype &obj, UINT index)
- const Etype & **operator[]** (UINT index) const
- Etype & **operator[]** (UINT index)
- bool **operator==** (const **Vector** &rhv)

Protected Member Functions

- int **min** (UINT left, UINT right) const
- void **verifyIndex** (UINT index) const

Protected Attributes

- UINT **m_size**
- UINT **m_capacity**
- UINT **m_increment**
- Etype ∗∗ **m_pData**

5.19.1 Detailed Description

template<class Etype> class Vector< Etype >

Implements a **Vector** (p. 96) class like Java class **Vector** (p. 96).

This class is the class **Vector** (p. 96) with templates.

Author:

Albert Orriols-Puig and Nuria Macia
Grup de Recerca en Sistemes Intel.ligents
La Salle - Universitat Ramon Llull
C/ Quatre Camins, 2. 08022, Barcelona (Spain)

Date:

Created April, 2009
Last modified December, 2010

5.19.2 Constructor & Destructor Documentation

5.19.2.1 template<class Etype> Vector< Etype >::Vector (UINT *initialCapacity* = `10`, UINT *capacityIncrement* = `10`) `[inline]`

Constructs a new **Vector** (p. 96) object.

Parameters:

initialCapacity is the initial capacity of the vector.

capacityIncrement is the amount by which the capacity is increased when the vector overflows.

References Vector< Etype >::m_capacity, Vector< Etype >::m_increment, Vector< Etype >::m_pData, and Vector< Etype >::m_size.

5.19.2.2 template<class Etype> Vector< Etype >::Vector (const Vector< Etype > & *rhv*) `[inline]`

Copy constructor.

Parameters:

rhv is the **Vector** (p. 96) object that will be copied to the new one.

References Vector< Etype >::m_capacity, Vector< Etype >::m_increment, Vector< Etype >::m_pData, and Vector< Etype >::m_size.

5.19.2.3 template<class Etype> Vector< Etype >::∼Vector () `[inline, virtual]`

Destructs a **Vector** (p. 96) object.

References Vector< Etype >::m_pData, and Vector< Etype >::removeAllElements().

5.19.3 Member Function Documentation

5.19.3.1 template<class Etype> UINT Vector< Etype >::capacity () const `[inline]`

Returns the capacity of the vector.

Returns:

An integer with the capacity of the vector.

References Vector< Etype >::m_capacity.

5.19.3.2 template<class Etype> bool Vector< Etype >::contains (const Etype & *elem*) const `[inline]`

Returns whether the **Vector** (p. 96) contains the passed element.

Parameters:

elem is the element that will be searched for.

Returns:

A Boolean that indicates whether the element has been found.

Return values:

true the element has been found.
false the element has not been found.

References Vector< Etype >::m_pData, and Vector< Etype >::m_size.

5.19.3.3 template<class Etype> const Etype & Vector< Etype >::firstElement () const `[inline]`

Returns the first element of the vector.

Returns:

The first element of the vector.

References Utils::logWarningError(), Vector< Etype >::m_pData, and Vector< Etype >::m_size.

5.19.3.4 template<class Etype> int Vector< Etype >::indexOf (const Etype & *elem*) const `[inline]`

Returns the index of a given element.

Parameters:

elem is the element that will be searched for.

Returns:

An integer that indicates the position where the element has been found.

Return values:

-1 the element has not been found.

References Vector< Etype >::m_pData, and Vector< Etype >::m_size.

5.19.3.5 template<class Etype> bool Vector< Etype >::isEmpty () const `[inline]`

Returns whether the **Vector** (p. 96) is empty or not.

Returns:

A Boolean that inidicates whether the vector is empty.

Return values:

true the vector is empty.
false the vector contains elements.

References Vector< Etype >::m_size.

5.19.3.6 template<class Etype> const Etype & Vector< Etype >::lastElement () const `[inline]`

Returns the last element.

Returns:

The last element.

References Vector< Etype >::m_pData, and Vector< Etype >::m_size.

5.19.3.7 template<class Etype> int Vector< Etype >::lastIndexOf (const Etype & *elem*) const `[inline]`

Returns the last index where a given element appears.

Parameters:

elem is the element that will be searched for.

Returns:

An integer with the last position where the element has been found.

References Vector< Etype >::m_pData, and Vector< Etype >::m_size.

5.19.3.8 template<class Etype> UINT Vector< Etype >::size () const `[inline]`

Returns the number of elements in the vector.

Returns:

An integer with the number of elements in the vector.

References Vector< Etype >::m_size.

5.19.3.9 template<class Etype> void Vector< Etype >::copyInto (Etype ∗ *array*) const `[inline]`

Copies the array into the vector.

Parameters:

array is the array to be copied.

References Vector< Etype >::m_pData, and Vector< Etype >::m_size.

5.19.3.10 template<class Etype> void Vector< Etype >::addElement (const Etype & *obj*) `[inline]`

Adds an element in the vector.

Parameters:

obj is the element to be added.

References Vector< Etype >::ensureCapacity(), Vector< Etype >::m_capacity, Vector< Etype >::m_increment, Vector< Etype >::m_pData, and Vector< Etype >::m_size.

5.19.3.11 template<class Etype> void Vector< Etype >::ensureCapacity (UINT *minCapacity*) `[inline]`

Ensures capacity by increasing it if necessary.

Parameters:

minCapacity is the minimum capacity.

References Vector< Etype >::m_capacity, Vector< Etype >::m_pData, and Vector< Etype >::m_size.

Referenced by Vector< Etype >::addElement(), Vector< Etype >::insertElementAt(), and Vector< Etype >::setSize().

5.19.3.12 template<class Etype> void Vector< Etype >::removeAllElements () `[inline]`

Removes all the elements from the vector.

References Vector< Etype >::m_pData, and Vector< Etype >::m_size.

Referenced by Vector< Etype >::∼Vector().

5.19.3.13 template<class Etype> bool Vector< Etype >::removeElement (const Etype & *obj*) `[inline]`

Removes the given element.

Parameters:

obj is the element to be removed.

Returns:

A Boolean that indicates whether the element has been removed.

Return values:

true the element has been removed successfully.

false the element has not been removed.

References Vector< Etype >::m_pData, Vector< Etype >::m_size, and Vector< Etype >::removeElementAt().

5.19.3.14 template<class Etype> void Vector< Etype >::setSize (UINT *newSize*) `[inline]`

Sets the size of the vector.

Parameters:

newSize is the new size of the vector.

References Vector< Etype >::ensureCapacity(), Vector< Etype >::m_capacity, Vector< Etype >::m_pData, and Vector< Etype >::m_size.

5.19.3.15 template<class Etype> void Vector< Etype >::trimToSize () `[inline]`

Trims the vector to the actual size.

References Vector< Etype >::m_capacity, Vector< Etype >::m_pData, and Vector< Etype >::m_size.

5.19.3.16 template<class Etype> Etype & Vector< Etype >::elementAt (UINT *index*) const `[inline]`

Returns the element at the indicated position.

Parameters:

index is the position to be consulted.

Returns:

The corresponding element.

References Vector< Etype >::m_pData, and Vector< Etype >::verifyIndex().

Referenced by Vector< Etype >::operator[]().

5.19.3.17 template<class Etype> void Vector< Etype >::insertElementAt (const Etype & *obj*, UINT *index*) [inline]

Inserts an element into a specified position.

Parameters:

obj is the object to be inserted.

index is the position where the object will be inserted.

References Vector< Etype >::ensureCapacity(), Vector< Etype >::m_capacity, Vector< Etype >::m_increment, Vector< Etype >::m_pData, Vector< Etype >::m_size, and Vector< Etype >::verifyIndex().

5.19.3.18 template<class Etype> void Vector< Etype >::removeElementAt (UINT *index*) [inline]

Removes an element from the specified position.

Parameters:

index is the position that will be removed.

References Vector< Etype >::m_pData, Vector< Etype >::m_size, and Vector< Etype >::verifyIndex().

Referenced by Vector< Etype >::removeElement().

5.19.3.19 template<class Etype> void Vector< Etype >::removeElementAtWithoutDestroyingElement (UINT *index*) [inline]

Removes an element from one position without deleting the element.

Parameters:

index is the position of the element that will be removed.

References Vector< Etype >::m_pData, Vector< Etype >::m_size, and Vector< Etype >::verifyIndex().

5.19.3.20 template<class Etype> void Vector< Etype >::setElementAt (const Etype & *obj*, UINT *index*) [inline]

Sets the element at a specified position.

Parameters:

obj is the object to be inserted.

index is the position where the object will be inserted.

References Vector< Etype >::m_pData, and Vector< Etype >::verifyIndex().

5.19.3.21 template<class Etype> const Etype & Vector< Etype >::operator[] (UINT *index*) const `[inline]`

Overloads the operator [].

Parameters:

index is the position of the consulted element.

Returns:

The consulted element.

References Vector< Etype >::elementAt().

5.19.3.22 template<class Etype> Etype & Vector< Etype >::operator[] (UINT *index*) `[inline]`

Overloads the operator [].

Parameters:

index is the position of the consulted element.

Returns:

The consulted element.

References Vector< Etype >::m_pData, and Vector< Etype >::verifyIndex().

5.19.3.23 template<class Etype> bool Vector< Etype >::operator== (const Vector< Etype > & *rhv*) `[inline]`

Overloads the operator ==.

Parameters:

rhv is the vector with which the current vector will be compared.

Returns:

A Boolean that indicates whether the two elements are equal.

Return values:

true the current vector is equal to rhv.

false the current vector is different from rhv.

References Vector< Etype >::m_pData, and Vector< Etype >::m_size.

5.19.3.24 template<class Etype> int Vector< Etype >::min (UINT *left*, UINT *right*) const `[inline, protected]`

Returns the minimum value between the two arguments.

Parameters:

left is the first argument.

right is the second argument.

Returns:

An integer with the minimum value between left and right.

5.19.3.25 template<class Etype> void Vector< Etype >::verifyIndex (UINT *index*) const `[inline, protected]`

Checks whether the index is correct.

Parameters:

index is the index to be checked.

References Vector< Etype >::m_capacity.

Referenced by Vector< Etype >::elementAt(), Vector< Etype >::insertElementAt(), Vector< Etype >::operator[]\(), Vector< Etype >::removeElementAt(), Vector< Etype >::removeElementAtWithoutDestroyingElement(), and Vector< Etype >::setElementAt().

5.19.4 Field Documentation

5.19.4.1 template<class Etype> UINT Vector< Etype >::m_size `[protected]`

Size of the vector.

Referenced by Vector< Etype >::addElement(), Vector< Etype >::contains(), Vector< Etype >::copyInto(), Vector< Etype >::ensureCapacity(), Vector< Etype >::firstElement(), Vector< Etype >::indexOf(), Vector< Etype >::insertElementAt(), Vector< Etype >::isEmpty(), Vector< Etype >::lastElement(), Vector< Etype >::lastIndexOf(), Vector< Etype >::operator==(), Vector< Etype >::removeAllElements(), Vector< Etype >::removeElement(), Vector< Etype >::removeElementAt(), Vector< Etype >::removeElementAtWithoutDestroyingElement(), Vector< Etype >::setSize(), Vector< Etype >::size(), Vector< Etype >::trimToSize(), and Vector< Etype >::Vector().

5.19.4.2 template<class Etype> UINT Vector< Etype >::m_capacity `[protected]`

Capacity of the vector.

Referenced by Vector< Etype >::addElement(), Vector< Etype >::capacity(), Vector< Etype >::ensureCapacity(), Vector< Etype >::insertElementAt(), Vector< Etype >::setSize(), Vector< Etype >::trimToSize(), Vector< Etype >::Vector(), and Vector< Etype >::verifyIndex().

5.19.4.3 template<class Etype> UINT Vector< Etype >::m_increment `[protected]`

Minimum increment when the vector runs out of memory.

Referenced by Vector< Etype >::addElement(), Vector< Etype >::insertElementAt(), and Vector< Etype >::Vector().

5.19.4.4 template<class Etype> Etype** Vector< Etype >::m_pData `[protected]`

Vector (p. 96) where the elements are stored.

Referenced by Vector< Etype >::addElement(), Vector< Etype >::contains(), Vector< Etype >::copyInto(), Vector< Etype >::elementAt(), Vector< Etype >::ensureCapacity(), Vector< Etype >::firstElement(), Vector< Etype >::indexOf(), Vector< Etype >::insertElementAt(), Vector< Etype >::lastElement(), Vector< Etype >::lastIndexOf(), Vector< Etype >::operator==(), Vector< Etype >::operator[](), Vector< Etype >::removeAllElements(), Vector< Etype >::removeElement(), Vector< Etype >::removeElementAt(), Vector< Etype >::removeElementAtWithoutDestroyingElement(), Vector< Etype >::setElementAt(), Vector< Etype >::setSize(), Vector< Etype >::trimToSize(), Vector< Etype >::Vector(), and Vector< Etype >::~Vector().

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

- Vector.h

Index