Reference Manual

1.0

Contents

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 2

Class Index

2.1 Class Hierarchy

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

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Namespace Documentation

5.1 pfe Namespace Reference

Classes

- class PatternClassifier
- class RFClassifier

  Random forest classifier.
- class VectorSpaceModel
- class Clusterer
- class Word
- class CorpusParser

  Class for reading corpus from PFE format into memory.
- struct CoverElement
- class Cover
- class CorpusIndex

  CorpusIndex class helps to calculate covers of different words in a corpus efficiently.
- struct ModelBuildSettings

  Class representing possible settings for building a model.
- class Model
- class RFModel
- class RecPrecModel
- class ClusterModel
- class ClusterFeatureModel
- class Automaton
- class Pattern

  Pattern consist of left context, entity area and right context, which are all automatons.
- class Criteria

  Base class for pattern mining criteria.
- class GoodPrecisionCriteria

  Criteria, that maximizes precision.
- class GoodRecallCriteria

  Criteria, that maximizer recall.
- class MixedCriteria

  Criteria, that tries sums the precision and recall as weight.
- class UniquenessCriteria

  Criteria, that tries to maximize uniqueness.
- class RandomGeneralizer

Base class for random generalizers for optimisation tasks.

- class RandomAttributeGeneralizer
- class StringIndex
- struct WordMeaning
- class Wordnet
- class WordnetParser

Typedefs

- typedef std::pair < CoverElement, double > CoverElementProb
- typedef std::vector < CoverElementProb > ElementProbVector
- typedef std::vector< CoverElement > CoverElementList
- typedef std::vector< Cover > CoverVector
- typedef std::vector< Word > Sentence
- typedef std::vector< Sentence > Corpus
- typedef std::vector< double > DoubleVector
- typedef std::vector< Automaton > AutomatonVector
- typedef std::vector< Word > WordVector
- typedef std::vector< unsigned > WordIndexVector
- typedef std::vector< Pattern > PatternVector
- typedef std::vector< std::string > StringVector

Enumerations

- enum LCATrim { LCATrimNone, LCATrimLeft, LCATrimRight }
- enum WordnetRelations { WORDNET_HAS_HYPERONYM, WORDNET_HAS_HYPONYM, WORDNET_BELONGS_TO_CLASS }
- enum WordnetPOS { WORDNET_NONE, WORDNET_VERB, WORDNET_NOUN, WORDNET_ADJECTIVE, WORDNET_ADVERB, WORDNET_PROPER_NOUN }

Functions

- Corpus mapToCorpus (Corpus const &corpus, ElementProbVector const &v, std::string const &annot, double treshold=0.5)
- Cover filterElements (ElementProbVector const &v, double treshold=0.5)
- std::vector< size_t > kmedoidsIndices (const Corpus &corpus, size_t const K)

  Simple clustering functions ///.
- std::vector< Corpus > kmedoids (const Corpus &corpus, size_t const K)

  Kmedoids clustering function that cluster given corpus into K smaller corpora.
- unsigned long getID (std::string const &attribute)

  Global function to get attribute string from attribute ID.
- unsigned long getID (const char *const attribute)
- std::string getString (unsigned long const ID)

  Global function to get string from attribute ID.
- std::unordered_map< unsigned long, size_t > getAnnotCounts (Corpus const &corpus)

  Get a unordered map of annotation counts from the corpus.
- std::string toString (Sentence::const_iterator begin, Sentence::const_iterator end)
- std::string toString (Sentence const &sentence, size_t begin, size_t end)
- std::string toString (Sentence const &sentence)

- Corpus randomCorpusSample (Corpus corpus, size_t k)

  Make a random sample of given corpus of size k.
- std::ostream & printCoverElementLemmata (CoverElement const &elem, Corpus const &corpus, std::ostream &os)
- std::ostream & printCoverElementOriginal (CoverElement const &elem, Corpus const &corpus, std::ostream &os)
- std::ostream & printCoverLemmata (Cover const &cover, Corpus const &corpus, std::ostream &os)
- std::ostream & printCoverOriginal (Cover const &cover, Corpus const &corpus, std::ostream &os)
- template<typename T >
  std::vector< T > set_union (std::vector< T > const &A, std::vector< T > const &B)
- template<typename T >
  std::vector< T > set_intersection (std::vector< T > const &A, std::vector< T > const &B)
- template<typename T >
  std::vector< T > set_difference (std::vector< T > const &A, std::vector< T > const &B)
- void renderPatterns (PatternVector const &patterns, std::string path, std::string title)
- void randomSearch (PatternVector const &patterns, CorpusIndex &index, Cover const &trueCover, size_t N, Criteria *criteria, RandomGeneralizer *generalizer, PatternVector &resPatterns, CoverVector &resCovers)
- void writeString (std::string const &s, std::ostream &os)
- std::string readString (std::istream &is)
- void writeDouble (double value, std::ostream &os)
- double readDouble (std::istream &is)
- void writeUl (unsigned long value, std::ostream &os)
- unsigned long readUl (std::istream &is)
- void writeLong (long value, std::ostream &os)
- long readLong (std::istream &is)
- int getNumAvailableCores ()
- std::size_t getNumThreads ()
- void setNumThreads (size_t n)
- void intFunction (int value)
- bool boolReturnFunction ()
- void split_sentences (std::string textf, std::string splitterf, bool signs=false)
- void split_sentences (std::istream &, std::ostream &, bool signs=false)
- std::ostream & operator<< (std::ostream &os, RFClassifier &classifier)
- std::istream & operator>> (std::istream &is, RFClassifier &classifier)
- std::unordered_map< unsigned
  long, double > getTfWeights (Sentence const &sentence)
- double getDistance (std::unordered_map< unsigned long, double > &idf1, std::unordered_map< unsigned long, double > &idf2)
- std::vector< size_t > initRandomMedoids (size_t K, size_t const N)

  Helper method for clusterer that creates random initial medoids.
- std::vector< std::vector
  < size_t > > divideToClusters (VectorSpaceModel &model, Corpus const &corpus, std::vector< size_t > &medoids)

  Helper method for clusterer for dividing sentences to clusters.
- size_t findBestMedoid (std::vector< size_t > indices, VectorSpaceModel &model)
- std::ostream & operator<< (std::ostream &os, Clusterer &clusterer)
- std::istream & operator>> (std::istream &is, Clusterer &clusterer)
- bool getBit (uintmax_t value, size_t pos)
- std::ostream & operator<< (std::ostream &os, Word const &word)
- std::istream & operator>> (std::istream &is, Word &word)
- std::ostream & operator<< (std::ostream &os, ModelBuildSettings &settings)
- std::ostream & operator<< (std::ostream &os, RFModel &model)
- std::istream & operator>> (std::istream &is, RFModel &model)
- PatternVector buildPart (Corpus &trainCorpus, Cover &trainCover, Corpus &testCorpus, Cover &testCover, ModelBuildSettings &settings)

- std::ostream & operator<< (std::ostream &os, RecPrecModel &model)
- std::istream & operator>> (std::istream &is, RecPrecModel &model)
- std::ostream & operator<< (std::ostream &os, ClusterModel &model)
- std::istream & operator>> (std::istream &is, ClusterModel &model)
- std::ostream & operator<< (std::ostream &os, ClusterFeatureModel &model)
- std::istream & operator>> (std::istream &is, ClusterFeatureModel &model)
- std::ostream & operator<< (std::ostream &os, Automaton const &automaton)
- std::istream & operator>> (std::istream &is, Automaton &automaton)
- void patternThreadMatcher (Pattern const &pattern, CorpusIndex &index, Cover &cover, int &completed)
- std::ostream & operator<< (std::ostream &os, Pattern const &pattern)
- std::istream & operator>> (std::istream &is, Pattern &pattern)
- PatternVector makeRandomCandidates (PatternVector const &patterns, RandomGeneralizer *generalizer, size_t const N)

Variables

- const size_t MAX_WORD_ATTRIBUTES = 32
- const size_t PFE_MAJOR_VERSION = 1

  Library version.
- const size_t PFE_MINOR_VERSION = 0
- size_t numThreads = 1

5.1.1 Detailed Description

This file is part of Pattern Based Fact Extraction library (PFE)

```
PFE is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

PFE is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

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

5.1.2 Typedef Documentation

5.1.2.1 typedef std::vector<Automaton> pfe::AutomatonVector

5.1.2.2 typedef std::vector<Sentence> pfe::Corpus

5.1.2.3 typedef std::vector<CoverElement> pfe::CoverElementList

5.1.2.4 typedef std::pair<CoverElement, double> pfe::CoverElementProb

5.1.2.5 typedef std::vector<Cover> pfe::CoverVector

5.1.2.6 typedef std::vector<double> pfe::DoubleVector

5.1.2.7 typedef std::vector<CoverElementProb> pfe::ElementProbVector

5.1.2.8 typedef std::vector<Pattern> pfe::PatternVector

5.1.2.9 typedef std::vector<Word> pfe::Sentence

5.1.2.10 typedef std::vector<std::string> pfe::StringVector

5.1.2.11 typedef std::vector<unsigned> pfe::WordIndexVector

5.1.2.12 typedef std::vector<Word> pfe::WordVector

5.1.3 Enumeration Type Documentation

5.1.3.1 enum pfe::LCATrim

Enumeration, that is used to tell if to trim automatons from left or right, if the lengths do not match. Trim NONE means that LCA operation should fail in case lengths do not match.

Enumerator:

LCATrimNone

LCATrimLeft

LCATrimRight

5.1.3.2 enum pfe::WordnetPOS

Enumerator:

WORDNET_NONE

WORDNET_VERB

WORDNET_NOUN

WORDNET_ADJECTIVE

WORDNET_ADVERB

WORDNET_PROPER_NOUN

5.1.3.3 enum pfe::WordnetRelations

Enumerator:

WORDNET_HAS_HYPERONYM

WORDNET_HAS_HYPONYM

WORDNET_BELONGS_TO_CLASS

5.1.4 Function Documentation

5.1.4.1 bool pfe::boolReturnFunction ( )

5.1.4.2 PatternVector pfe::buildPart (Corpus & *trainCorpus,* Cover & *trainCover,* Corpus & *testCorpus,* Cover & *testCover,* ModelBuildSettings & *settings*)

5.1.4.3 std::vector<std::vector<size_t> > pfe::divideToClusters (VectorSpaceModel & *model,* Corpus const & *corpus,* std::vector< size_t > & *medoids*)

Helper method for clusterer for dividing sentences to clusters.

5.1.4.4 Cover pfe::filterElements (ElementProbVector const & *v,* double *treshold* = 0.5)

5.1.4.5 size_t pfe::findBestMedoid (std::vector< size_t > *indices,* VectorSpaceModel & *model*)

5.1.4.6 std::unordered_map< unsigned long, size_t > pfe::getAnnotCounts (Corpus const & *corpus*)

Get a unordered map of annotation counts from the corpus.

5.1.4.7 bool pfe::getBit (uintmax_t *value,* size_t *pos*)

5.1.4.8 double pfe::getDistance (std::unordered_map< unsigned long, double > & *idf1,* std::unordered_map< unsigned long, double > & *idf2*)

5.1.4.9 unsigned long pfe::getID (std::string const & *attribute*)

Global function to get attribute string from attribute ID.

5.1.4.10 unsigned long pfe::getID (const char *const *attribute*)

5.1.4.11 int pfe::getNumAvailableCores ( )

Almost platform independent way (win,macos,linux) to determine the number of available cpus. Taken from: http://stackoverflow.com/questions/150355/programmatically-find-the-number-of-cores-on-a-mac

The reason using this is that std::thread::hardware_concurrency() does not work always.

5.1.4.12 size_t pfe::getNumThreads ( )

Get the number of threads the program should use for parallel computations.

5.1.4.13 std::string pfe::getString (unsigned long const *ID*)

Global function to get string from attribute ID.

5.1.4.14 std::unordered_map<unsigned long, double> pfe::getTfWeights (Sentence const & *sentence*)

5.1.4.15 std::vector<size_t> pfe::initRandomMedoids (size_t *K,* size_t const *N*)

Helper method for clusterer that creates random initial medoids.

5.1.4.16 void pfe::intFunction (int *value*)

5.1.4.17 std::vector< Corpus > pfe::kmedoids (const Corpus & *corpus,* size_t const *K*)

Kmedoids clustering function that cluster given corpus into K smaller corpora.

5.1.4.18 std::vector< size_t > pfe::kmedoidsIndices (const Corpus & *corpus,* size_t const *K*)

Simple clustering functions ///.

Kmedoids clustering function that cluster given corpus into K smaller corpora. The function returns a vector of sentence indices, which defines the cluster for each sentence in the given corpus.

5.1.4.19 **PatternVector** pfe::makeRandomCandidates (PatternVector const & *patterns,* RandomGeneralizer * *generalizer,* size_t const *N*)

5.1.4.20 **Corpus** pfe::mapToCorpus (Corpus const & *corpus,* ElementProbVector const & *v,* std::string const & *annot,* double *treshold* = 0.5)

5.1.4.21 std::ostream& pfe::operator<< (std::ostream & *os,* ModelBuildSettings & *settings*)

5.1.4.22 std::ostream& pfe::operator<< (std::ostream & *os,* RFClassifier & *classifier*)

5.1.4.23 std::ostream& pfe::operator<< (std::ostream & *os,* RFModel & *model*)

5.1.4.24 std::ostream& pfe::operator<< (std::ostream & *os,* Clusterer & *clusterer*)

5.1.4.25 std::ostream& pfe::operator<< (std::ostream & *os,* Word const & *word*)

5.1.4.26 std::ostream& pfe::operator<< (std::ostream & *os,* RecPrecModel & *model*)

5.1.4.27 std::ostream& pfe::operator<< (std::ostream & *os,* Automaton const & *automaton*)

5.1.4.28 std::ostream& pfe::operator<< (std::ostream & *os,* ClusterModel & *model*)

5.1.4.29 std::ostream& pfe::operator<< (std::ostream & *os,* ClusterFeatureModel & *model*)

5.1.4.30 std::ostream& pfe::operator<< (std::ostream & *os,* Pattern const & *pattern*)

5.1.4.31 std::istream& pfe::operator>> (std::istream & *is,* RFClassifier & *classifier*)

5.1.4.32 std::istream& pfe::operator>> (std::istream & *is,* RFModel & *model*)

5.1.4.33 std::istream& pfe::operator>> (std::istream & *is,* Word & *word*)

5.1.4.34 std::istream& pfe::operator>> (std::istream & *is,* Clusterer & *clusterer*)

5.1.4.35 std::istream& pfe::operator>> (std::istream & *is,* RecPrecModel & *model*)

5.1.4.36 std::istream& pfe::operator>> (std::istream & *is,* Automaton & *automaton*)

5.1.4.37 std::istream& pfe::operator>> (std::istream & *is,* ClusterModel & *model*)

5.1.4.38 std::istream& pfe::operator>> (std::istream & *is,* ClusterFeatureModel & *model*)

5.1.4.39 std::istream& pfe::operator>> (std::istream & *is,* Pattern & *pattern*)

5.1.4.40 void pfe::patternThreadMatcher (Pattern const & *pattern,* CorpusIndex & *index,* Cover & *cover,* int & *completed*)

5.1.4.41 std::ostream& pfe::printCoverElementLemmata (CoverElement const & *elem,* Corpus const & *corpus,* std::ostream & *os*) [inline]

5.1.4.42 std::ostream& pfe::printCoverElementOriginal (CoverElement const & *elem,* Corpus const & *corpus,* std::ostream & *os*) [inline]

5.1.4.43 std::ostream& pfe::printCoverLemmata (Cover const & *cover,* Corpus const & *corpus,* std::ostream & *os*) [inline]

Function for printing the cover contents as represented lemmata.

5.1.4.44 std::ostream& pfe::printCoverOriginal (Cover const & *cover,* Corpus const & *corpus,* std::ostream & *os*) [inline]

Function for printing the cover as represented original contents.

5.1.4.45 Corpus pfe::randomCorpusSample (Corpus *corpus,* size_t *k*)

Make a random sample of given corpus of size k.

5.1.4.46 void pfe::randomSearch (PatternVector const & *patterns,* CorpusIndex & *index,* Cover const & *trueCover,* size_t *N,* Criteria * *criteria,* RandomGeneralizer * *generalizer,* PatternVector & *resPatterns,* CoverVector & *resCovers*)

5.1.4.47 double pfe::readDouble (std::istream & *is*)

5.1.4.48 long pfe::readLong (std::istream & *is*)

5.1.4.49 std::string pfe::readString (std::istream & *is*)

5.1.4.50 unsigned long pfe::readUl (std::istream & *is*)

5.1.4.51 void pfe::renderPatterns (PatternVector const & *patterns,* std::string *path,* std::string *title*)

5.1.4.52 template<typename T > std::vector<T> pfe::set_difference (std::vector< T > const & *A,* std::vector< T > const & *B*)

Set difference operator on generic vectors. We do not use STL version of the function, because that does not allow const arguments.

5.1.4.53 template<typename T > std::vector<T> pfe::set_intersection (std::vector< T > const & *A,* std::vector< T > const & *B*)

Set intersection operator on generic vectors. We do not use STL version of the function, because that does not allow const arguments.

5.1.4.54 template<typename T > std::vector<T> pfe::set_union (std::vector< T > const & *A,* std::vector< T > const & *B*)

Set union operator on generic vectors. We do not use STL version of the function, because that does not allow const arguments.

5.1.4.55 void pfe::setNumThreads (size_t *n*)

Set the number of threads the program should use for parallel computations.

5.1.4.56 void pfe::split_sentences (std::string *textf,* std::string *splitterf,* bool *signs* = false)

5.1.4.57 void pfe::split_sentences (std::istream & *textstream,* std::ostream & *splitterstream,* bool *signs* = false)

5.1.4.58 std::string pfe::toString (Sentence::const_iterator *begin,* Sentence::const_iterator *end*)

5.1.4.59 std::string pfe::toString (Sentence const & *sentence,* size_t *begin,* size_t *end*)

5.1.4.60 std::string pfe::toString (Sentence const & *sentence*)

5.1.4.61 void pfe::writeDouble (double *value,* std::ostream & *os*)

5.1.4.62 void pfe::writeLong (long *value,* std::ostream & *os*)

5.1.4.63 void pfe::writeString (std::string const & *s,* std::ostream & *os*)

5.1.4.64 void pfe::writeUl (unsigned long *value,* std::ostream & *os*)

5.1.5 Variable Documentation

5.1.5.1 const size_t pfe::MAX_WORD_ATTRIBUTES = 32

Maximal number of attributes for a word. We define it to let the compiler do some optimizations. Try to keep this as low as possible as this is the main thing, that takes the memory.

5.1.5.2 size_t pfe::numThreads = 1

5.1.5.3 const size_t pfe::PFE_MAJOR_VERSION = 1

Library version.

5.1.5.4 const size_t pfe::PFE_MINOR_VERSION = 0

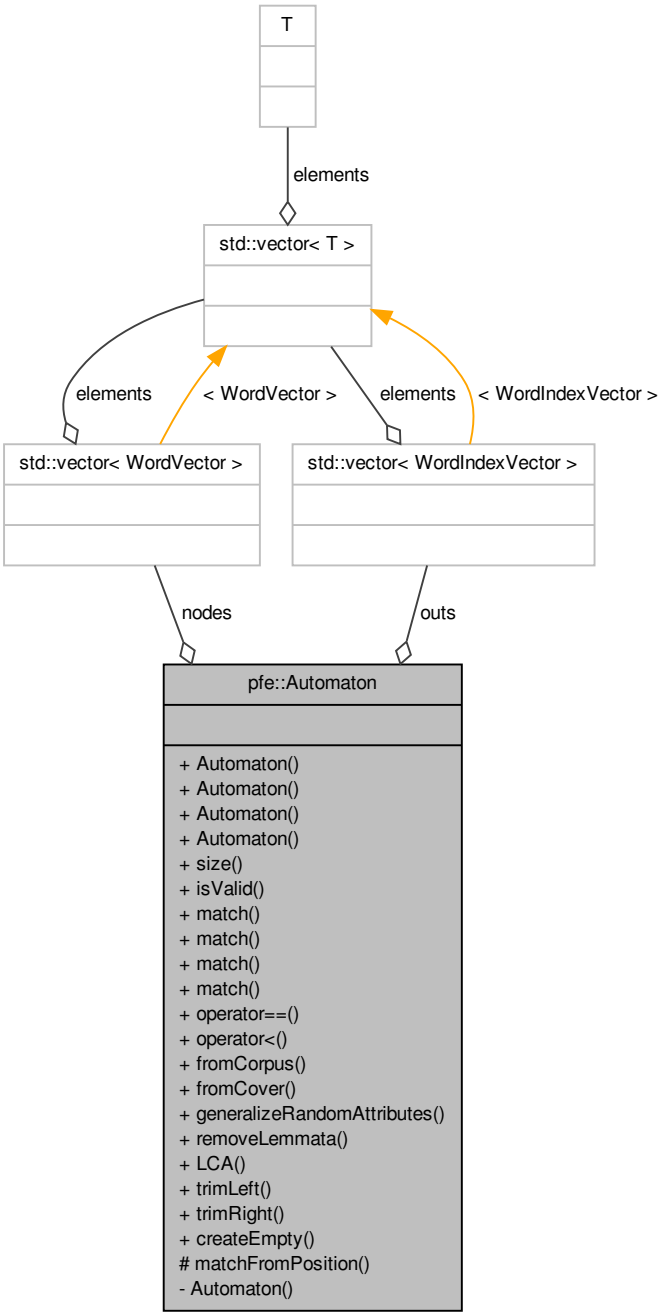
Chapter 6

Class Documentation

6.1 pfe::Automaton Class Reference

```
#include <Pattern.hpp>
```

Collaboration diagram for pfe::Automaton:



Public Member Functions

- Automaton (Sentence::const_iterator begin, Sentence::const_iterator end)
- Automaton (Sentence const &sentence, unsigned const begin, unsigned const end)
- Automaton (Automaton const &A)
- Automaton (std::vector< WordVector > const &nodes, std::vector< WordIndexVector > const &outs)
- size_t size () const

- bool isValid () const
- Cover match (Sentence const &sentence, unsigned sentenceID) const
- Cover match (Corpus const &corpus, const std::vector< unsigned > &sampleIndices, bool allSentences=false) const
- Cover match (Corpus const &corpus) const
- Cover match (CorpusIndex &index) const
- bool operator== (Automaton const &other) const
- bool operator< (Automaton const &other) const

Static Public Member Functions

- static AutomatonVector fromCorpus (Corpus const &corpus, unsigned long const annotation)

  Create automatons from corpus. O(n) function.
- static AutomatonVector fromCover (Corpus const &corpus, Cover const &cover)
- static Automaton generalizeRandomAttributes (Automaton const &automaton)
- static Automaton removeLemmata (Automaton const &automaton)
- static Automaton LCA (Automaton const &A, Automaton const &B, LCATrim trimming=LCATrimNone)
- static Automaton trimLeft (Automaton const &A, size_t N)
- static Automaton trimRight (Automaton const &A, size_t N)
- static Automaton createEmpty ()

Protected Member Functions

- Cover matchFromPosition (Sentence const &sentence, unsigned sentenceID, unsigned wordIdx) const

Protected Attributes

- std::vector< WordVector > nodes
- std::vector< WordIndexVector > outs

Private Member Functions

- Automaton ()

Friends

- class Pattern
- std::ostream & operator<< (std::ostream &os, Automaton const &automaton)
- std::istream & operator>> (std::istream &is, Automaton &automaton)

6.1.1 Constructor & Destructor Documentation

6.1.1.1 pfe::Automaton::Automaton ( ) [inline], [private]

6.1.1.2 pfe::Automaton::Automaton (Sentence::const_iterator *begin*, Sentence::const_iterator *end*)

6.1.1.3 pfe::Automaton::Automaton (Sentence const & *sentence*, unsigned const *begin*, unsigned const *end*)

6.1.1.4 pfe::Automaton::Automaton (Automaton const & *A*) [inline]

6.1.1.5 pfe::Automaton::Automaton (std::vector< WordVector > const & *nodes,* std::vector< WordIndexVector > const & *outs*) [inline]

6.1.2 Member Function Documentation

6.1.2.1 Automaton pfe::Automaton::createEmpty ( ) [static]

6.1.2.2 AutomatonVector pfe::Automaton::fromCorpus (Corpus const & *corpus,* unsigned long const *annotation*) [static]

Create automatons from corpus. O(n) function.

6.1.2.3 AutomatonVector pfe::Automaton::fromCover (Corpus const & *corpus,* Cover const & *cover*) [static]

6.1.2.4 Automaton pfe::Automaton::generalizeRandomAttributes (Automaton const & *automaton*) [static]

6.1.2.5 bool pfe::Automaton::isValid ( ) const

6.1.2.6 Automaton pfe::Automaton::LCA (Automaton const & *A,* Automaton const & *B,* LCATrim *trimming* = LCATrimNone) [static]

6.1.2.7 Cover pfe::Automaton::match (Sentence const & *sentence,* unsigned *sentenceID*) const

6.1.2.8 Cover pfe::Automaton::match (Corpus const & *corpus,* const std::vector< unsigned > & *sampleIndices,* bool *allSentences* = false) const

6.1.2.9 Cover pfe::Automaton::match (Corpus const & *corpus*) const

6.1.2.10 Cover pfe::Automaton::match (CorpusIndex & *index*) const

6.1.2.11 Cover pfe::Automaton::matchFromPosition (Sentence const & *sentence,* unsigned *sentenceID,* unsigned *wordIdx*) const [protected]

6.1.2.12 bool pfe::Automaton::operator< (Automaton const & *other*) const

6.1.2.13 bool pfe::Automaton::operator== (Automaton const & *other*) const

6.1.2.14 Automaton pfe::Automaton::removeLemmata (Automaton const & *automaton*) [static]

6.1.2.15 size_t pfe::Automaton::size ( ) const [inline]

6.1.2.16 Automaton pfe::Automaton::trimLeft (Automaton const & *A,* size_t *N*) [static]

6.1.2.17 Automaton pfe::Automaton::trimRight (Automaton const & *A,* size_t *N*) [static]

6.1.3 Friends And Related Function Documentation

6.1.3.1 std::ostream& operator<< (std::ostream & *os,* Automaton const & *automaton*) [friend]

6.1.3.2 std::istream& operator>> (std::istream & *is,* Automaton & *automaton*) [friend]

6.1.3.3 friend class Pattern [friend]

6.1.4 Member Data Documentation

6.1.4.1 std::vector<WordVector> pfe::Automaton::nodes [protected]

6.1.4.2 std::vector<WordIndexVector> pfe::Automaton::outs [protected]

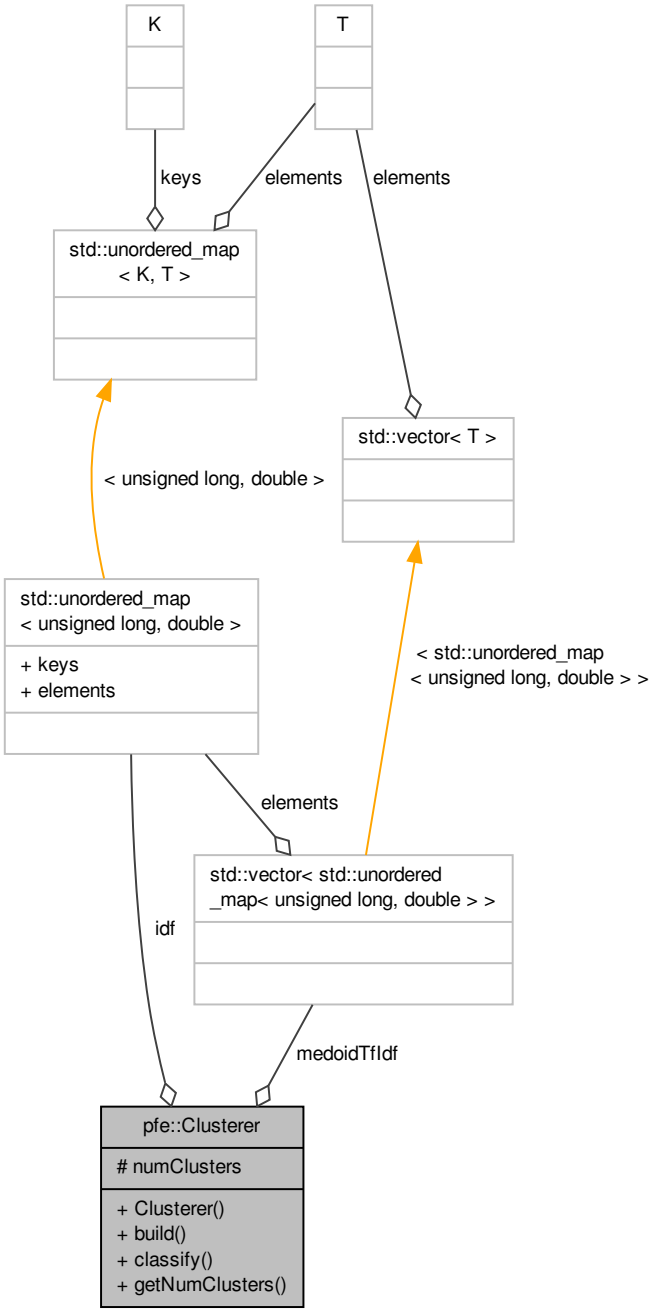
The documentation for this class was generated from the following files:

- include/Pattern.hpp
- src/Pattern.cpp

6.2 pfe::Clusterer Class Reference

```
#include <Clustering.hpp>
```

Collaboration diagram for pfe::Clusterer:



Public Member Functions

- Clusterer ()
- std::vector< std::vector < size_t > > build (Corpus const &corpus, size_t const numClusters)
- size_t classify (Sentence const &sentence)
- size_t getNumClusters () const

Protected Attributes

- std::vector < std::unordered_map< unsigned long, double > > medoidTfIdf
- std::unordered_map< unsigned long, double > idf

  idf weights for words from training corpus.
- size_t numClusters

Friends

- std::ostream & operator<< (std::ostream &os, Clusterer &clusterer)
- std::istream & operator>> (std::istream &is, Clusterer &clusterer)

6.2.1 Detailed Description

Class implementing kmedoids clustering on a corpus using VectorSpaceModel.

6.2.2 Constructor & Destructor Documentation

6.2.2.1 pfe::Clusterer::Clusterer ( ) [inline]

6.2.3 Member Function Documentation

6.2.3.1 std::vector< std::vector< size_t > > pfe::Clusterer::build (Corpus const & *corpus*, size_t const *numClusters*)

It also returns clusters as each element of the vector containing the sentence indices in the original corpus.

6.2.3.2 size_t pfe::Clusterer::classify (Sentence const & *sentence*)

6.2.3.3 size_t pfe::Clusterer::getNumClusters ( ) const [inline]

6.2.4 Friends And Related Function Documentation

6.2.4.1 std::ostream& operator<< (std::ostream & *os*, Clusterer & *clusterer*) [friend]

6.2.4.2 std::istream& operator>> (std::istream & *is*, Clusterer & *clusterer*) [friend]

6.2.5 Member Data Documentation

6.2.5.1 std::unordered_map<unsigned long, double> pfe::Clusterer::idf [protected]

idf weights for words from training corpus.

6.2.5.2 std::vector<std::unordered_map<unsigned long, double> > pfe::Clusterer::medoidTfIdf [protected]

6.2.5.3 size_t pfe::Clusterer::numClusters [protected]

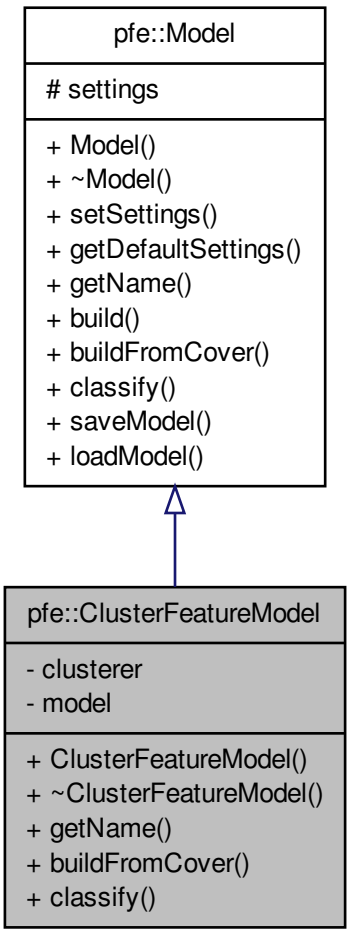
The documentation for this class was generated from the following files:

- include/Clustering.hpp
- src/Clustering.cpp

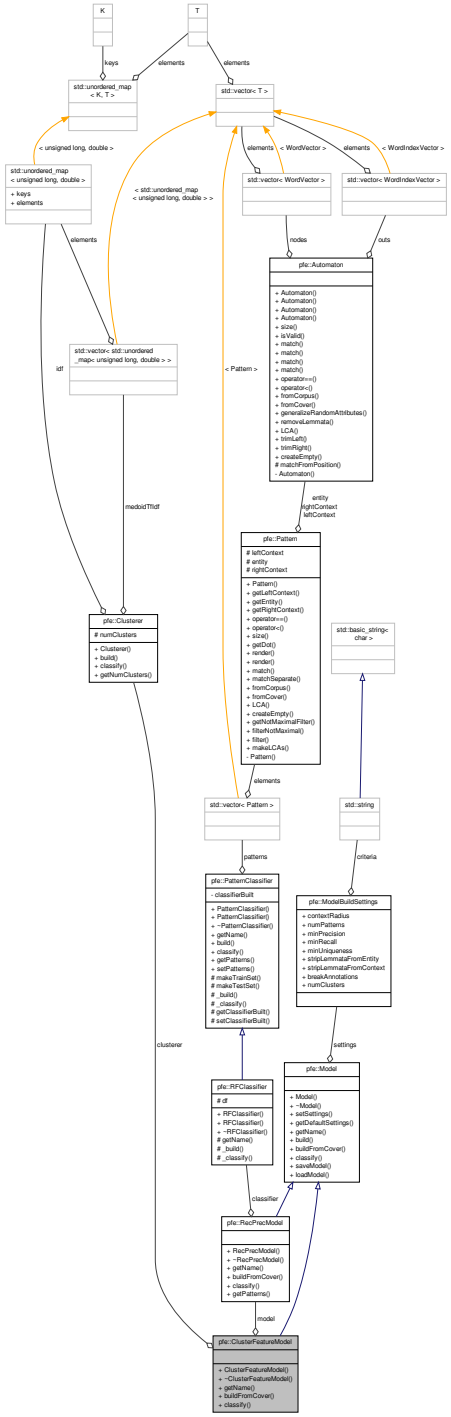
6.3 pfe::ClusterFeatureModel Class Reference

`#include <Model.hpp>`

Inheritance diagram for pfe::ClusterFeatureModel:



Collaboration diagram for pfe::ClusterFeatureModel:



Public Member Functions

- ClusterFeatureModel ()
- virtual ~ClusterFeatureModel ()
- virtual std::string getName () const
- virtual void buildFromCover (Corpus const &corpus, Cover const &trueCover)
- virtual ElementProbVector classify (Corpus const &corpus, CorpusIndex &index)

Private Attributes

- Clusterer clusterer
- RecPrecModel model

Friends

- std::ostream & operator<< (std::ostream &os, ClusterFeatureModel &model)
- std::istream & operator>> (std::istream &is, ClusterFeatureModel &model)

Additional Inherited Members

6.3.1 Detailed Description

ClusterFeatureModel. Adds clustering as a feature for RecPrecModel

6.3.2 Constructor & Destructor Documentation

6.3.2.1 pfe::ClusterFeatureModel::ClusterFeatureModel ( ) [inline]

6.3.2.2 virtual pfe::ClusterFeatureModel::~ClusterFeatureModel ( ) [inline], [virtual]

6.3.3 Member Function Documentation

6.3.3.1 void pfe::ClusterFeatureModel::buildFromCover (Corpus const & *corpus*, Cover const & *trueCover*) [virtual]

Implements pfe::Model.

6.3.3.2 ElementProbVector pfe::ClusterFeatureModel::classify (Corpus const & *corpus*, CorpusIndex & *index*) [virtual]

Implements pfe::Model.

6.3.3.3 virtual std::string pfe::ClusterFeatureModel::getName ( ) const [inline], [virtual]

Implements pfe::Model.

6.3.4 Friends And Related Function Documentation

6.3.4.1 std::ostream& operator<< (std::ostream & *os*, ClusterFeatureModel & *model*) [friend]

6.3.4.2 std::istream& operator>> (std::istream & *is*, ClusterFeatureModel & *model*) [friend]

6.3.5 Member Data Documentation

6.3.5.1 Clusterer pfe::ClusterFeatureModel::clusterer [private]

6.3.5.2 RecPrecModel pfe::ClusterFeatureModel::model [private]

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

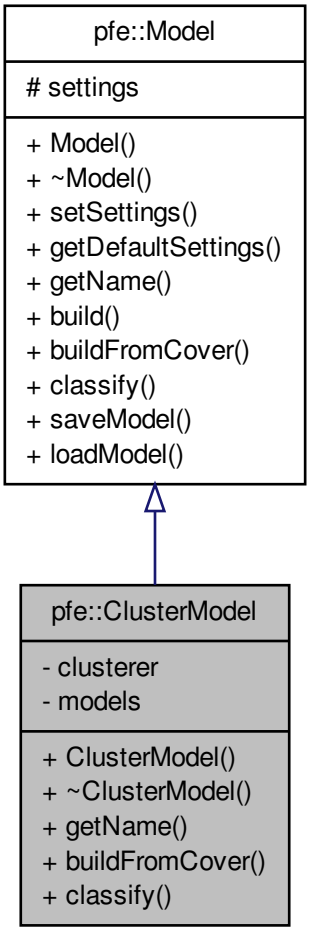
- include/Model.hpp

- src/Model.cpp

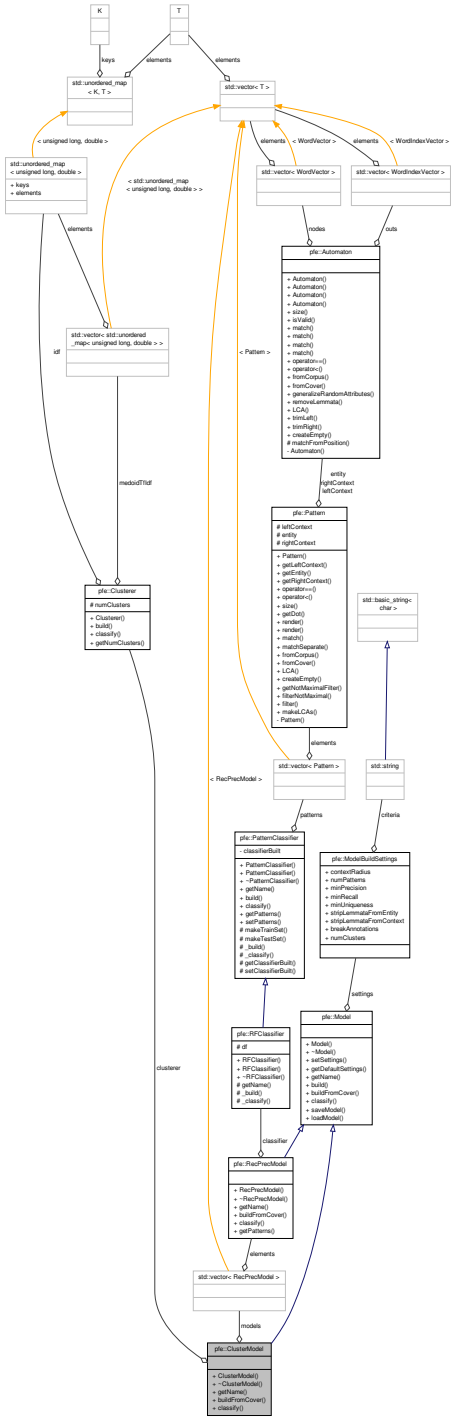
6.4 pfe::ClusterModel Class Reference

`#include <Model.hpp>`

Inheritance diagram for pfe::ClusterModel:



Collaboration diagram for pfe::ClusterModel:



Public Member Functions

- ClusterModel ()
- virtual ~ClusterModel ()
- virtual std::string getName () const
- virtual void buildFromCover (Corpus const &corpus, Cover const &trueCover)
- virtual ElementProbVector classify (Corpus const &corpus, CorpusIndex &index)

Private Attributes

- Clusterer clusterer
- std::vector< RecPrecModel > models

Friends

- std::ostream & operator<< (std::ostream &os, ClusterModel &model)
- std::istream & operator>> (std::istream &is, ClusterModel &model)

Additional Inherited Members

6.4.1 Detailed Description

ClusterModel. Tries to cluster the sentences and train a RecPrecModel on each cluster. Classification first identifies, which cluster should be used

6.4.2 Constructor & Destructor Documentation

6.4.2.1 pfe::ClusterModel::ClusterModel ( ) [inline]

6.4.2.2 virtual pfe::ClusterModel::~ClusterModel ( ) [inline], [virtual]

6.4.3 Member Function Documentation

6.4.3.1 void pfe::ClusterModel::buildFromCover (Corpus const & *corpus,* Cover const & *trueCover*) [virtual]

Implements pfe::Model.

6.4.3.2 ElementProbVector pfe::ClusterModel::classify (Corpus const & *corpus,* CorpusIndex & *index*) [virtual]

Implements pfe::Model.

6.4.3.3 virtual std::string pfe::ClusterModel::getName ( ) const [inline], [virtual]

Implements pfe::Model.

6.4.4 Friends And Related Function Documentation

6.4.4.1 std::ostream& operator<< (std::ostream & *os,* ClusterModel & *model*) [friend]

6.4.4.2 std::istream& operator>> (std::istream & *is,* ClusterModel & *model*) [friend]

6.4.5 Member Data Documentation

6.4.5.1 Clusterer pfe::ClusterModel::clusterer [private]

6.4.5.2 std::vector<RecPrecModel> pfe::ClusterModel::models [private]

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

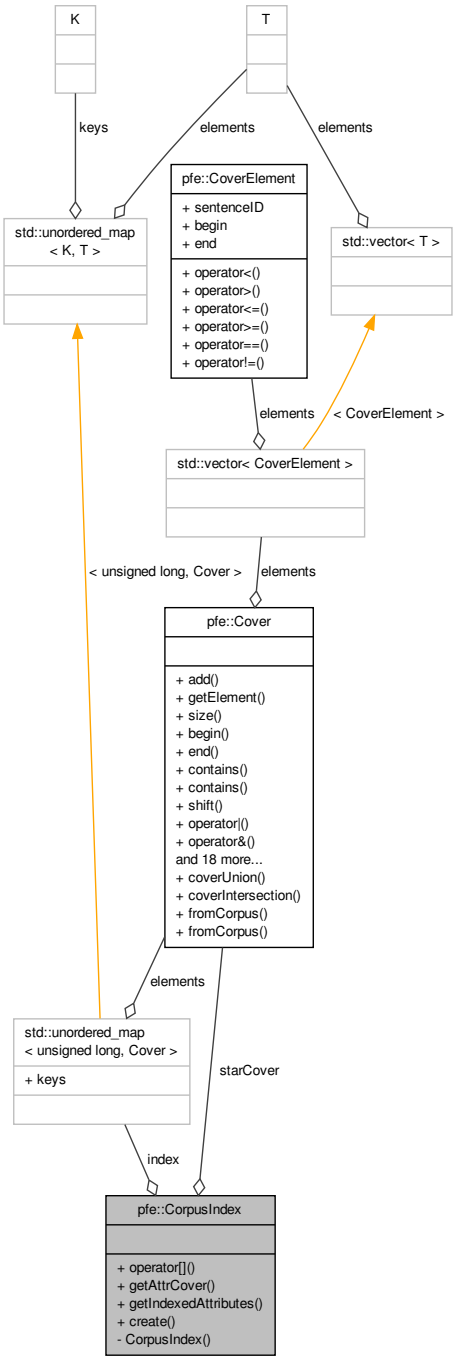
- include/Model.hpp
- src/Model.cpp

6.5 pfe::CorpusIndex Class Reference

CorpusIndex class helps to calculate covers of different words in a corpus efficiently.

`#include <Corpus.hpp>`

Collaboration diagram for pfe::CorpusIndex:



Public Member Functions

- Cover operator[] (Word const &word)

  Overloaded operator, that results the Cover of given word using the underlying index.
- Cover getAttrCover (unsigned long attr)

  Get the cover of a single attribute. This is equivalent of using word with only that attribute.
- std::vector< unsigned long > getIndexedAttributes ()

Static Public Member Functions

- static CorpusIndex create (Corpus const &corpus)

  Method, that creates an index from the given corpus.

Private Member Functions

- CorpusIndex ()

  Private constructor to prohibit explicit initialization of the index.

Private Attributes

- std::unordered_map< unsigned long, Cover > index

  The internal attribute ID to Cover index.
- Cover starCover

  The cover of the most generic word matching every position in the corpus.

6.5.1 Detailed Description

CorpusIndex class helps to calculate covers of different words in a corpus efficiently.

6.5.2 Constructor & Destructor Documentation

6.5.2.1 pfe::CorpusIndex::CorpusIndex ( ) [inline], [private]

Private constructor to prohibit explicit initialization of the index.

6.5.3 Member Function Documentation

6.5.3.1 CorpusIndex pfe::CorpusIndex::create (Corpus const & *corpus*) [static]

Method, that creates an index from the given corpus.

6.5.3.2 Cover pfe::CorpusIndex::getAttrCover (unsigned long *attr*)

Get the cover of a single attribute. This is equivalent of using word with only that attribute.

6.5.3.3 std::vector< unsigned long > pfe::CorpusIndex::getIndexedAttributes ( )

6.5.3.4 Cover pfe::CorpusIndex::operator[] (Word const & *word*)

Overloaded operator, that results the Cover of given word using the underlying index.

6.5.4 Member Data Documentation

6.5.4.1 std::unordered_map<unsigned long, Cover> pfe::CorpusIndex::index [private]

The internal attribute ID to Cover index.

6.5.4.2 Cover pfe::CorpusIndex::starCover [private]

The cover of the most generic word matching every position in the corpus.

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

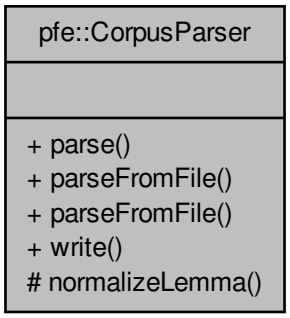
- include/Corpus.hpp
- src/Corpus.cpp

6.6 pfe::CorpusParser Class Reference

Class for reading corpus from PFE format into memory.

`#include <Corpus.hpp>`

Collaboration diagram for pfe::CorpusParser:



Public Member Functions

- Corpus parse (std::istream &is, boost::function< void(int)>=boost::bind(&intFunction, _1), boost::function< bool(void)>=boost::bind(&boolReturnFunction))

  Parse a corpus from a given input stream.
- Corpus parseFromFile (std::string const &filename, boost::function< void(int)> progressCallback=boost::bind(&intFunction, _1), boost::function< bool(void)> cancelCallback=boost::bind(&boolReturnFunction))

  Parse a corpus from a file determined by given filename.
- Corpus parseFromFile (const char *const filename, boost::function< void(int)> progressCallback=boost::bind(&intFunction, _1), boost::function< bool(void)> cancelCallback=boost::bind(&boolReturnFunction))

  Parse a corpus from a file determined by given filename.
- void write (Corpus const &corpus, std::ostream &os)

Protected Member Functions

- std::string normalizeLemma (std::string &lemma)

  Given a lemma with additinal notation, keep only the canonical form.

6.6.1 Detailed Description

Class for reading corpus from PFE format into memory.

6.6.2 Member Function Documentation

6.6.2.1 std::string pfe::CorpusParser::normalizeLemma (std::string & *lemma*) [protected]

Given a lemma with additinal notation, keep only the canonical form.

6.6.2.2 Corpus pfe::CorpusParser::parse (std::istream & *is*, boost::function< void(int)> *progressCallback* = boost::bind(&intFunction, _1), boost::function< bool(void)> *cancelCallback* = boost::bind(&boolReturnFunction))

Parse a corpus from a given input stream.

6.6.2.3 Corpus pfe::CorpusParser::parseFromFile (std::string const & *filename*, boost::function< void(int)> *progressCallback* = boost::bind(&intFunction, _1), boost::function< bool(void)> *cancelCallback* = boost::bind(&boolReturnFunction))

Parse a corpus from a file determined by given filename.

6.6.2.4 Corpus pfe::CorpusParser::parseFromFile (const char *const *filename*, boost::function< void(int)> *progressCallback* = boost::bind(&intFunction, _1), boost::function< bool(void)> *cancelCallback* = boost::bind(&boolReturnFunction))

Parse a corpus from a file determined by given filename.

6.6.2.5 void pfe::CorpusParser::write (Corpus const & *corpus*, std::ostream & *os*)

Write the loaded corpus back to specified stream. NB! Each word must have lemma set (not equal to ID 0).

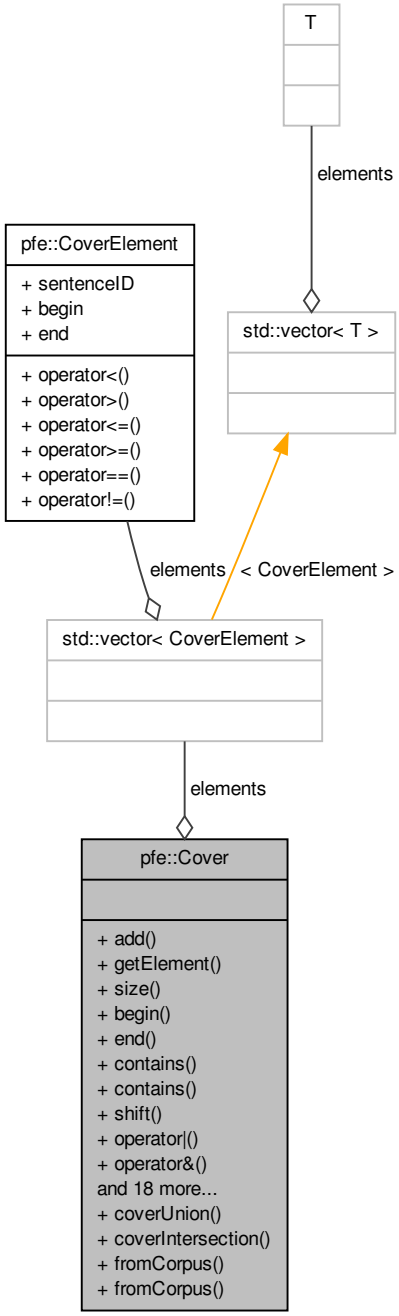
The documentation for this class was generated from the following files:

- include/Corpus.hpp
- src/Corpus.cpp

6.7 pfe::Cover Class Reference

```
#include <Corpus.hpp>
```

Collaboration diagram for pfe::Cover:



Public Member Functions

- void add (CoverElement const &elem, bool const toEnd=false)

  Add a CoverElement to Cover. If toEnd is true, then put the element directly to end.
- CoverElement getElement (size_t const pos) const

  Get the CoverElement, that is indexed by value of pos.
- size_t size () const

Get the number of CoverElement instances in this cover.

- CoverElementList::const_iterator begin () const

  Constant iterator to beginning of the CoverElement vector.

- CoverElementList::const_iterator end () const

  Constant iterator to end of the CoverElement vector.

- bool contains (CoverElement const &elem) const

  Does the cove contain given element.

- bool contains (Cover const &cover) const

  Does the cover contain another cover.

- Cover shift (int pos, Corpus const &corpus)
- Cover operator| (Cover const &other) const

  Union operator of the two covers. Resulting cover has elements from both this and the other cover.

- Cover operator& (Cover const &other) const

  Intersect operator of the two covers. Resulting cover has elements only in both covers.

- Cover operator+ (Cover const &other) const
- Cover operator- (Cover const &other) const

  Cover difference operator. Resulting cover has all elements that are in this cover.

- bool operator== (Cover const &other) const

  Equality operator.

- Cover applyLeftContext (Cover const &leftContext) const
- Cover applyRightContext (Cover const &rightContext) const
- Cover applyLeftRightContext (Cover const &leftContext, Cover const &rightContext) const
- Cover removeInner () const

  Remove inner annotations.

- bool doesOverlap (CoverElement const &elem) const
- Cover breakDown () const
- void renderAttribute (Corpus &corpus, unsigned long attribute) const

  Given a corpus, add given attribute to all sentence fragments covered by this cover.

- size_t numFp (Cover const &trueCover) const
- double fprate (Cover const &trueCover) const
- double fnrate (Cover const &trueCover) const
- double precision (Cover const &trueCover) const
- double recall (Cover const &trueCover) const
- double fscore (Cover const &trueCover) const
- double uniqueness (Cover const &other) const

  Estimate of how unique is this cover compared to the other cover: $|cover - other| / |cover|$.

- double avgUniqueness (CoverVector const &others) const

Static Public Member Functions

- static Cover coverUnion (CoverVector const &covers)

  Static member functio that returns the union of all given covers.

- static Cover coverIntersection (CoverVector const &covers)

  Static member function that returns the intersection of all given covers.

- static Cover fromCorpus (Corpus const &corpus, unsigned long const annotation)

  Create a cover instance from given corpus from consequent words having given annotation.

- static Cover fromCorpus (Corpus const &corpus)

  Create a full cover from corpus, that includes all 1-element entries.

Private Attributes

- CoverElementList elements

  The vector containing the ordered CoverElement instances.

Friends

- std::ostream & operator<< (std::ostream &os, Cover const &cover)

  Friend function that prints the representation of this cover to the stream.
- std::ostream & printCoverElementLemmata (CoverElement const &elem, Corpus const &corpus, std::ostream &os)
- std::ostream & printCoverElementOriginal (CoverElement const &elem, Corpus const &corpus, std::ostream &os)
- std::ostream & printCoverLemmata (Cover const &cover, Corpus const &corpus, std::ostream &os)
- std::ostream & printCoverOriginal (Cover const &cover, Corpus const &corpus, std::ostream &os)

6.7.1 Detailed Description

Cover represents a set of CoverElement structures.

Cover is a central structure in PFE, which is used from matching patterns from an index and calculating statistical estimations. It stores CoverElement structures in a sorted vector for minimal memory usage and fast union, intersect and concatenate operations.

6.7.2 Member Function Documentation

6.7.2.1 void pfe::Cover::add (CoverElement const & *elem,* bool const *toEnd* = `false`)

Add a CoverElement to Cover. If toEnd is true, then put the element directly to end.

6.7.2.2 Cover pfe::Cover::applyLeftContext (Cover const & *leftContext*) const

Given the cover representing the left context of some pattern, the resulting cover contains only elements, that would be used in concatenation process from any element in left cover.

6.7.2.3 Cover pfe::Cover::applyLeftRightContext (Cover const & *leftContext,* Cover const & *rightContext*) const

Given the left and right context covers, keep only elements in result that would be used in concatenation both from left and right contet covers.

6.7.2.4 Cover pfe::Cover::applyRightContext (Cover const & *rightContext*) const

Given the cover representing the left context of some pattern, the resulting cover contains only elements, that would be used in concatenation process to any element in right context cover.

6.7.2.5 double pfe::Cover::avgUniqueness (CoverVector const & *others*) const

6.7.2.6 CoverElementList::const_iterator pfe::Cover::begin ( ) const

Constant iterator to beginning of the CoverElement vector.

6.7.2.7 Cover pfe::Cover::breakDown ( ) const

Break all CoverElements to length 1 CoverElements. For example element (1,2,4) would be broken to (1,2,3) and (1,3,4). It will preserve the cover element ordering invariant.

6.7.2.8 bool pfe::Cover::contains (CoverElement const & *elem*) const

Does the cove contain given element.

6.7.2.9 bool pfe::Cover::contains (Cover const & *cover*) const

Does the cover contain another cover.

6.7.2.10 Cover pfe::Cover::coverIntersection (CoverVector const & *covers*) [static]

Static member function that returns the intersection of all given covers.

6.7.2.11 Cover pfe::Cover::coverUnion (CoverVector const & *covers*) [static]

Static member functio that returns the union of all given covers.

6.7.2.12 bool pfe::Cover::doesOverlap (CoverElement const & *elem*) const

Does any element of the cover overlap the given coverelement, that is if the given coverelement starts and ends within on of the covers elements

6.7.2.13 CoverElementList::const_iterator pfe::Cover::end ( ) const

Constant iterator to end of the CoverElement vector.

6.7.2.14 double pfe::Cover::fnrate (Cover const & *trueCover*) const

6.7.2.15 double pfe::Cover::fprate (Cover const & *trueCover*) const

6.7.2.16 Cover pfe::Cover::fromCorpus (Corpus const & *corpus,* unsigned long const *annotation*) [static]

Create a cover instance from given corpus from consequent words having given annotation.

6.7.2.17 Cover pfe::Cover::fromCorpus (Corpus const & *corpus*) [static]

Create a full cover from corpus, that includes all 1-element entries.

6.7.2.18 double pfe::Cover::fscore (Cover const & *trueCover*) const

6.7.2.19 CoverElement pfe::Cover::getElement (size_t const *pos*) const

Get the CoverElement, that is indexed by value of pos.

6.7.2.20 size_t pfe::Cover::numFp (Cover const & *trueCover*) const

6.7.2.21 Cover pfe::Cover::operator& (Cover const & *other*) const

Intersect operator of the two covers. Resulting cover has elements only in both covers.

6.7.2.22 Cover pfe::Cover::operator+ (Cover const & *other*) const

Concatenation operator. Resulting cover has matches that start in this cover and end in the other cover. For example, if A = {(1,1,2), (1,3,4)} and B = {(1,3,5), (1,6,8)} then resulting cover is C = {(1,1,5)}

6.7.2.23 Cover pfe::Cover::operator- (Cover const & *other*) const

Cover difference operator. Resulting cover has all elements that are in this cover.

6.7.2.24 bool pfe::Cover::operator== (Cover const & *other*) const [inline]

Equality operator.

6.7.2.25 Cover pfe::Cover::operator| (Cover const & *other*) const

Union operator of the two covers. Resulting cover has elements from both this and the other cover.

6.7.2.26 double pfe::Cover::precision (Cover const & *trueCover*) const

6.7.2.27 double pfe::Cover::recall (Cover const & *trueCover*) const

6.7.2.28 Cover pfe::Cover::removeInner ( ) const

Remove inner annotations.

6.7.2.29 void pfe::Cover::renderAttribute (Corpus & *corpus,* unsigned long *attribute*) const

Given a corpus, add given attribute to all sentence fragments covered by this cover.

6.7.2.30 Cover pfe::Cover::shift (int *pos,* Corpus const & *corpus*)

Shift the cover elements in the sentence by given number of positions. Note that elements that become invalid, are omitted.

6.7.2.31 size_t pfe::Cover::size ( ) const

Get the number of CoverElement instances in this cover.

6.7.2.32 double pfe::Cover::uniqueness (Cover const & *other*) const

Estimate of how unique is this cover compared to the other cover: |cover - other| / |cover|.

6.7.3 Friends And Related Function Documentation

6.7.3.1 std::ostream& operator<< (std::ostream & *os,* Cover const & *cover*) [friend]

Friend function that prints the representation of this cover to the stream.

6.7.3.2 std::ostream& printCoverElementLemmata (CoverElement const & *elem,* Corpus const & *corpus,* std::ostream & *os*) [friend]

6.7.3.3 std::ostream& printCoverElementOriginal (CoverElement const & *elem,* Corpus const & *corpus,* std::ostream & *os*) [friend]

6.7.3.4 std::ostream& printCoverLemmata (Cover const & *cover,* Corpus const & *corpus,* std::ostream & *os*) [friend]

Function for printing the cover contents as represented lemmata.

6.7.3.5 std::ostream& printCoverOriginal (Cover const & *cover,* Corpus const & *corpus,* std::ostream & *os*) [friend]

Function for printing the cover as represented original contents.

6.7.4 Member Data Documentation

6.7.4.1 CoverElementList pfe::Cover::elements [private]

The vector containing the ordered CoverElement instances.

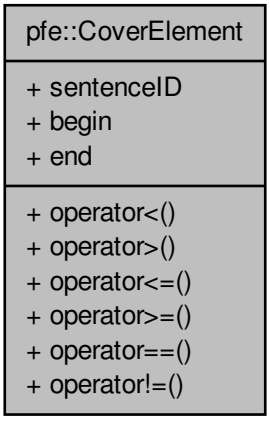
The documentation for this class was generated from the following files:

- include/Corpus.hpp
- src/Corpus.cpp

6.8 pfe::CoverElement Struct Reference

```
#include <Corpus.hpp>
```

Collaboration diagram for pfe::CoverElement:



Public Member Functions

- bool operator< (CoverElement const &other) const
- bool operator> (CoverElement const &other) const
- bool operator<= (CoverElement const &other) const
- bool operator>= (CoverElement const &other) const
- bool operator== (CoverElement const &other) const
- bool operator!= (CoverElement const &other) const

Public Attributes

- unsigned sentenceID

  The sentence index the match is.
- unsigned begin

  The position in sentence the match starts.
- unsigned end

  The position+1 in the sentence the match has ended.

Friends

- std::ostream & operator<< (std::ostream &os, CoverElement const &elem)

  Member function for printing the CoverElement representation to a stream.

6.8.1 Detailed Description

Coverelement specifies a single match or annotation.

Coverelement contains the sentence ID (index), the start position in the sentence and end position in the sentence (excluded).

6.8.2 Member Function Documentation

6.8.2.1 bool pfe::CoverElement::operator!= (CoverElement const & *other*) const

6.8.2.2 bool pfe::CoverElement::operator< (CoverElement const & *other*) const

6.8.2.3 bool pfe::CoverElement::operator<= (CoverElement const & *other*) const

6.8.2.4 bool pfe::CoverElement::operator== (CoverElement const & *other*) const

6.8.2.5 bool pfe::CoverElement::operator> (CoverElement const & *other*) const

6.8.2.6 bool pfe::CoverElement::operator>= (CoverElement const & *other*) const

6.8.3 Friends And Related Function Documentation

6.8.3.1 std::ostream& operator<< (std::ostream & *os,* CoverElement const & *elem*) [friend]

Member function for printing the CoverElement representation to a stream.

6.8.4 Member Data Documentation

6.8.4.1 unsigned pfe::CoverElement::begin

The position in sentence the match starts.

6.8.4.2 unsigned pfe::CoverElement::end

The position+1 in the sentence the match has ended.

6.8.4.3 unsigned pfe::CoverElement::sentenceID

The sentence index the match is.

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

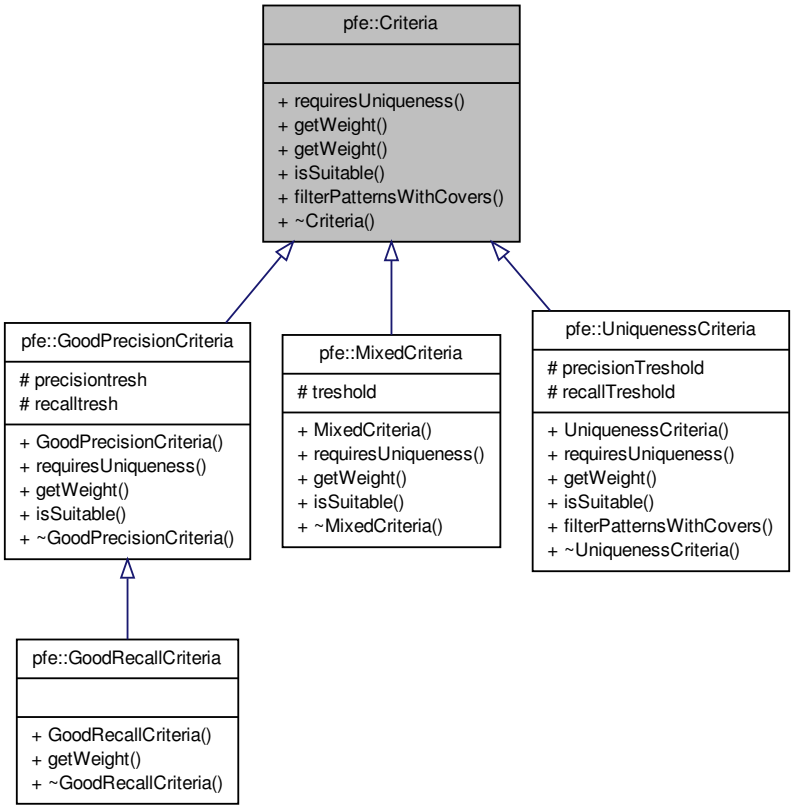
- include/Corpus.hpp
- src/Corpus.cpp

6.9 pfe::Criteria Class Reference

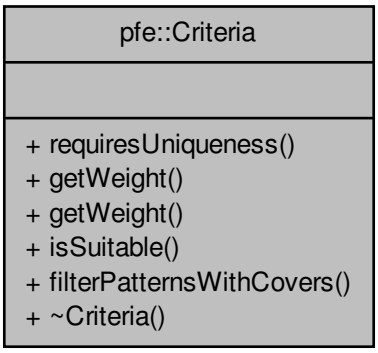
Base class for pattern mining criteria.

`#include <Pattern.hpp>`

Inheritance diagram for pfe::Criteria:



Collaboration diagram for pfe::Criteria:



Public Member Functions

- virtual bool requiresUniqueness () const =0
- virtual double getWeight (double precision, double recall)
- virtual double getWeight (double precision, double recall, double uniqueness)
- virtual bool isSuitable (double precision, double recall)
- virtual DoubleVector filterPatternsWithCovers (PatternVector &patterns, CoverVector &covers, Cover const &trueCover)
- virtual ~Criteria ()

6.9.1 Detailed Description

Base class for pattern mining criteria.

6.9.2 Constructor & Destructor Documentation

6.9.2.1 virtual pfe::Criteria::~Criteria ( ) [inline], [virtual]

6.9.3 Member Function Documentation

6.9.3.1 DoubleVector pfe::Criteria::filterPatternsWithCovers (PatternVector & *patterns,* CoverVector & *covers,* Cover const & *trueCover*) [virtual]

Reimplemented in pfe::UniquenessCriteria.

6.9.3.2 double pfe::Criteria::getWeight (double *precision,* double *recall*) [virtual]

Reimplemented in pfe::MixedCriteria, pfe::GoodRecallCriteria, and pfe::GoodPrecisionCriteria.

6.9.3.3 double pfe::Criteria::getWeight (double *precision,* double *recall,* double *uniqueness*) [virtual]

Reimplemented in pfe::UniquenessCriteria.

6.9.3.4 bool pfe::Criteria::isSuitable (double *precision,* double *recall*) [virtual]

Reimplemented in pfe::UniquenessCriteria, pfe::MixedCriteria, and pfe::GoodPrecisionCriteria.

6.9.3.5 virtual bool pfe::Criteria::requiresUniqueness ( ) const [pure virtual]

Implemented in pfe::UniquenessCriteria, pfe::MixedCriteria, and pfe::GoodPrecisionCriteria.

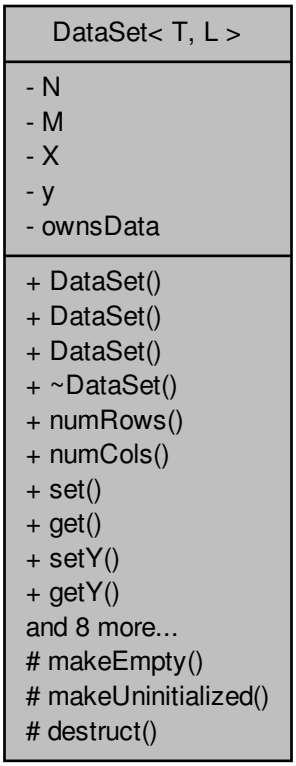
The documentation for this class was generated from the following files:

- include/Pattern.hpp
- src/Pattern.cpp

6.10 DataSet< T, L > Class Template Reference

```
#include <dfpar.hpp>
```

Collaboration diagram for DataSet< T, L >:



Public Member Functions

- DataSet ()
- DataSet (size_t const N, size_t const M)
- DataSet (size_t const N, size_t const M, T const *X, L const *y, bool ownsData=false)
- ~DataSet ()
- size_t numRows () const
- size_t numCols () const
- void set (size_t const row, size_t const col, T const value)
- double get (size_t const row, size_t const col) const
- void setY (size_t const row, L const &value)
- L getY (size_t const row) const
- size_t getIndex (size_t const row, size_t const col) const
- T * getFeatureMatrix () const
- void setFeatureMatrix (const T *X)
- L * getResponseVector () const
- void setResponseVector (const L *y)
- bool getOwnsData () const
- void setOwnsData (bool const ownsData)
- void createTrainAndTest (DataSet< T, L > &train, DataSet< T, L > &test, double proportion=0.8)

Protected Member Functions

- void makeEmpty ()
- void makeUninitialized (size_t const N, size_t const M)
- void destruct ()

Private Attributes

- size_t N
- size_t M
- T * X
- L * y
- bool ownsData

Friends

- std::ostream & operator<< (std::ostream &os, DataSet< T, L > const &ds)
- std::istream & operator>> (std::istream &is, DataSet< T, L > &ds)

6.10.1 Detailed Description

template<typename T, typename L>class DataSet< T, L >

Dataset class used by RFPAR library.

Template Parameters

T	Typename for feature data.
L	Typename for response (label) data.

6.10.2 Constructor & Destructor Documentation

6.10.2.1 template<typename T, typename L> DataSet< T, L >::DataSet ( ) [inline]

Construct an empty dataset. No memory is acquired for underlying data.

6.10.2.2 template<typename T, typename L> DataSet< T, L >::DataSet (size_t const *N*, size_t const *M*) [inline]

Create a dataset of size N*M. Method acquires memory for feature matrix as well as for response matrix. The values are left uninitialized, unless typenames T and L have default constructors, which is not the case for ordinary int, float and double types.

Parameters

N	the number of rows in the initialized dataset.
M	the number of columns in the initialized dataset.

6.10.2.3 template<typename T, typename L> DataSet< T, L >::DataSet (size_t const *N*, size_t const *M*, T const * *X*, L const * *y*, bool *ownsData* = false) [inline]

Create a dataset of size N*M.

Parameters

N	the number of rows in the initialized dataset.
M	the number of columns in the initialized dataset.
X	Pointer to feature matrix data.
y	Pointer to response vector data.
ownsData	If true, then releases the pointer to feature and response data.

6.10.2.4 template<typename T, typename L> DataSet< T, L >::~DataSet ( ) [inline]

Destructor. If ownsData is set true, releases memory for feature and response data.

6.10.3 Member Function Documentation

6.10.3.1 template<typename T, typename L> void DataSet< T, L >::createTrainAndTest (DataSet< T, L > & *train*, DataSet< T, L > & *test*, double *proportion* = 0.8) [inline]

Method from creating random test and train datasets.

Parameters

test	Uninitialized reference for train dataset.
train	Uninitialised reference for test dataset.
proportion	Value in range (0,1) determining how large proportion should be in trainset.

6.10.3.2 template<typename T, typename L> void DataSet< T, L >::destruct ( ) [inline], [protected]

Free the memory used by this DataSet instance, given that the instance owns the memory.

6.10.3.3 template<typename T, typename L> double DataSet< T, L >::get (size_t const *row*, size_t const *col*) const [inline]

Get the copy of the value at given row and col of the matrix.

Parameters

row	The given row index.
col	The given column index.

Returns

The copy of value at given row and column in feature matrix.

6.10.3.4 template<typename T, typename L> T* DataSet< T, L >::getFeatureMatrix ( ) const [inline]

Get the pointer to internal feature matrix of size N*M. Data is stored in a single array of size N*M with following structure: row_1: elem_1, elem_2, elem_3, ..., elem_M row_2: elem_{M+1}, .., elem{2M} ... row_N: ... elem{NM}

that is, it is stored row by row, starting from the first. Note that the above example used 1-based inidices, whereas in code you use 0-based indices.

Returns

The pointer to internal feature matrix.

6.10.3.5 template<typename T, typename L> size_t DataSet< T, L >::getIndex (size_t const *row*, size_t const *col*) const [inline]

Get the index of the element in data for given row and col indices.

Parameters

row	The given row index.
col	The given col index.

Returns

The element index in internal data pointer.

6.10.3.6 template<typename T, typename L> bool DataSet< T, L >::getOwnsData ( ) const [inline]

Returns

true, if the DataSet instance is responsible for releasing the memory upon destruction.

6.10.3.7 template<typename T, typename L> L* DataSet< T, L >::getResponseVector ( ) const [inline]

Returns

Pointer to underlying response vector.

6.10.3.8 template<typename T, typename L> L DataSet< T, L >::getY (size_t const *row*) const [inline]

Get the response value assigned to given row index.

Parameters

row	The given row index.

Returns

The assigned response value.

6.10.3.9 template<typename T, typename L> void DataSet< T, L >::makeEmpty ( ) [inline], [protected]

Create an empty dataset. No memory is acquired for underlying data.

6.10.3.10 template<typename T, typename L> void DataSet< T, L >::makeUninitialized (size_t const *N*, size_t const *M*) [inline], [protected]

Create a dataset of size N*M. Method acquires memory for feature matrix as well as for response matrix. The values are left uninitialized, unless typenames T and L have default constructors, which is not the case for ordinary int, float and double types.

Parameters

N	the number of rows in the initialized dataset.
M	the number of columns in the initialized dataset.

6.10.3.11 template<typename T, typename L> size_t DataSet< T, L >::numCols ( ) const [inline]

Returns

The number of columns in the dataset.

6.10.3.12 template<typename T, typename L> size_t DataSet< T, L >::numRows ( ) const [inline]

Returns

The number of rows in the dataset.

6.10.3.13 template<typename T, typename L> void DataSet< T, L >::set (size_t const *row*, size_t const *col*, T const *value*) [inline]

Set the matrix element at row and column to given value.

Parameters

row	Given row index.
col	Given column index.
The	value to assign to the feature matrix cell at (row, col).

6.10.3.14 template<typename T, typename L> void DataSet< T, L >::setFeatureMatrix (const T * *X*) [inline]

Replace the internal data array of the dataset.

The caller must ensure that data represents N rows and M columns. Also, if this DataSet instance is the owner of the data, it tries to free the memory of that pointer upon destruction.

Parameters

data	The pointer to new data. Data is not copied!

6.10.3.15 template<typename T, typename L> void DataSet< T, L >::setOwnsData (bool const *ownsData*) [inline]

Tell the DataSet instance, if it is responsible for releasing the memory upon destruction.

6.10.3.16 template<typename T, typename L> void DataSet< T, L >::setResponseVector (const L * *y*) [inline]

Replace the internal response vector of the dataset. Also, if this DataSet instance is the owner of the data, it tries to free the memory of that spointer upon destruction.

Parameters

y	The pointer to assign. Response values are not copied!

6.10.3.17 template<typename T, typename L> void DataSet< T, L >::setY (size_t const *row*, L const & *value*) [inline]

Set the response value for given row.

Parameters

row	The given row index.
value	The response value to assign to given row.

6.10.4 Friends And Related Function Documentation

6.10.4.1 template<typename T, typename L> std::ostream& operator<< (std::ostream & *os,* DataSet< T, L > const & *ds*) [friend]

Dataset serialization operator. The user must know the type of features and response values.

Parameters

os	The output stream.
dataset	The dataset.

Returns

The given output stream os.

6.10.4.2 template<typename T, typename L> std::istream& operator>> (std::istream & *is,* DataSet< T, L > & *ds*) [friend]

Dataset deserialization operator. The user must know the type of features and response values.

Parameters

is	The input stream.
dataset	The dataset.

Returns

The given input stream is.

6.10.5 Member Data Documentation

6.10.5.1 template<typename T, typename L> size_t DataSet< T, L >::M [private]

6.10.5.2 template<typename T, typename L> size_t DataSet< T, L >::N [private]

6.10.5.3 template<typename T, typename L> bool DataSet< T, L >::ownsData [private]

6.10.5.4 template<typename T, typename L> T* DataSet< T, L >::X [private]

6.10.5.5 template<typename T, typename L> L* DataSet< T, L >::y [private]

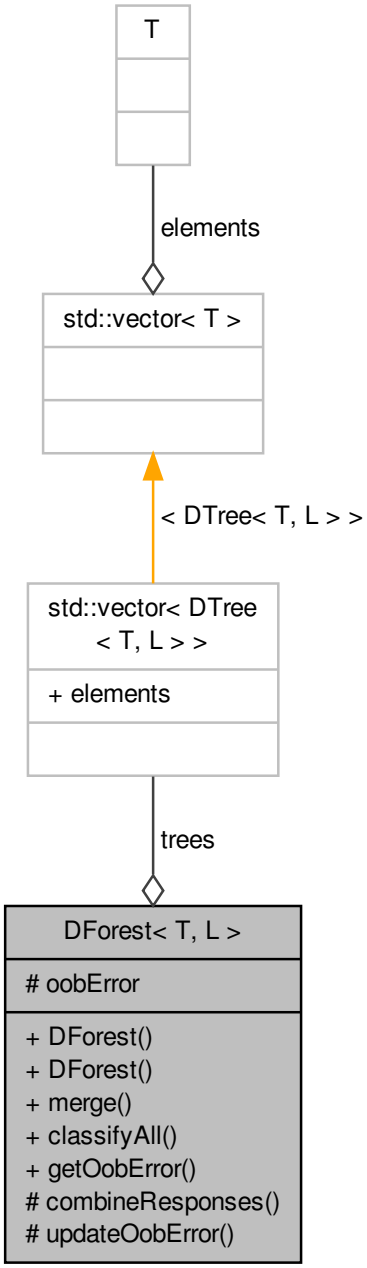
The documentation for this class was generated from the following file:

- include/dfpar.hpp

6.11 DForest< T, L > Class Template Reference

```
#include <dfpar.hpp>
```

Collaboration diagram for DForest< T, L >:



Public Member Functions

- DForest ()
- DForest (std::vector< DTree< T, L >> const &trees)
- void merge (DForest< T, L > const &otherForest)
- std::vector< ResultRow< L > > classifyAll (DataSet< T, L > const &ds, size_t const numThreads)
- double getOobError () const

Protected Member Functions

- std::vector< ResultRow< L > > combineResponses (DataSet< T, L > const &ds, std::vector< std::vector< L >> const &responses)
- void updateOobError ()

Protected Attributes

- std::vector< DTree< T, L > > trees

  Internal trees of the forest.
- double oobError

  Estimated out-of-bag error for the forest.

Friends

- class DForestBuilder< T, L >
- std::ostream & operator<< (std::ostream &os, DForest< T, L > &forest)
- std::istream & operator>> (std::istream &is, DForest< T, L > &forest)

6.11.1 Detailed Description

template<typename T, typename L>class DForest< T, L >

Decision forest class.

Template Parameters

T	Typename for feature data.
L	Typename for response (label) data.

6.11.2 Constructor & Destructor Documentation

6.11.2.1 template<typename T, typename L> DForest< T, L >::DForest ( ) [inline]

Initiate an empty decision forest instance.

6.11.2.2 template<typename T, typename L> DForest< T, L >::DForest (std::vector< DTree< T, L >> const & *trees*) [inline]

Initiate a decision forest instance from given list of trees.

6.11.3 Member Function Documentation

6.11.3.1 template<typename T, typename L> std::vector<ResultRow<L> > DForest< T, L >::classifyAll (DataSet< T, L > const & *ds*, size_t const *numThreads*) [inline]

Classify all rows in the dataset.

Parameters

ds	The dataset.
numThreads	How many threads to use for dividing up the work.

6.11.3.2 template<typename T, typename L> std::vector<ResultRow<L> > DForest< T, L >::combineResponses (DataSet< T, L > const & *ds*, std::vector< std::vector< L >> const & *responses*) [inline], [protected]

Method combining individual tree classifications into predicted labels and probabilities.

Parameters

ds	The dataset.
responses	vector of tree responses for all dataset rows.

6.11.3.3 template<typename T, typename L> double DForest< T, L >::getOobError ( ) const [inline]

Get the out-of-bag error estimate for the decision forest.

6.11.3.4 template<typename T, typename L> void DForest< T, L >::merge (DForest< T, L > const & *otherForest*) [inline]

Merge the trees of otherForest to this tree.

6.11.3.5 template<typename T, typename L> void DForest< T, L >::updateOobError ( ) [inline], [protected]

Update the decision forest oob error estimate.

6.11.4 Friends And Related Function Documentation

6.11.4.1 template<typename T, typename L> friend class DForestBuilder< T, L > [friend]

6.11.4.2 template<typename T, typename L> std::ostream& operator<< (std::ostream & *os*, DForest< T, L > & *forest*) [friend]

6.11.4.3 template<typename T, typename L> std::istream& operator>> (std::istream & *is*, DForest< T, L > & *forest*) [friend]

6.11.5 Member Data Documentation

6.11.5.1 template<typename T, typename L> double DForest< T, L >::oobError [protected]

Estimated out-of-bag error for the forest.

6.11.5.2 template<typename T, typename L> std::vector<DTree<T, L> > DForest< T, L >::trees [protected]

Internal trees of the forest.

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

- include/dfpar.hpp

6.12 DForestBuilder< T, L > Class Template Reference

`#include <dfpar.hpp>`

Collaboration diagram for DForestBuilder< T, L >:

Public Member Functions

- DForestBuilder (DataSet< T, L > const &ds)
- size_t getBootStrapSize () const
- void setBoostStrapSize (size_t const bootStrapSize)
- size_t getNumTrees () const
- void setNumTrees (size_t const numTrees)
- size_t getNumRandomFeatures () const
- void setNumRandomFeatures (size_t const numRandomFeatures)
- DForest< T, L > build (size_t numThreads=1)

Protected Member Functions

- void buildSingleTree (DTree< T, L > &tree)

Protected Attributes

- DataSet< T, L > const & ds
- size_t bootStrapSize
- size_t numTrees
- size_t numRandomFeatures

6.12.1 Detailed Description

template<typename T, typename L>class DForestBuilder< T, L >

Class for training decision forests.

Template Parameters

T	Typename for feature data.
L	Typename for response (label) data.

6.12.2 Constructor & Destructor Documentation

6.12.2.1 template<typename T , typename L > DForestBuilder< T, L >::DForestBuilder (DataSet< T, L > const & *ds*) [inline]

Initialize a decision forest builder.

Parameters

ds	The dataset to use for training.

6.12.3 Member Function Documentation

6.12.3.1 template<typename T , typename L > DForest<T, L> DForestBuilder< T, L >::build (size_t *numThreads* = 1) [inline]

Compute the decision forest.

Parameters

numThreads	The number of threads to use for training the model.

6.12.3.2 template<typename T , typename L > void DForestBuilder< T, L >::buildSingleTree (DTree< T, L > & *tree*) [inline], [protected]

6.12.3.3 template<typename T , typename L > size_t DForestBuilder< T, L >::getBootStrapSize ( ) const [inline]

Returns

The size of bootstrap sample used in training.

6.12.3.4 template<typename T , typename L > size_t DForestBuilder< T, L >::getNumRandomFeatures ( ) const [inline]

6.12.3.5 template<typename T , typename L > size_t DForestBuilder< T, L >::getNumTrees ( ) const [inline]

Returns

The number of trees to train for the decision forest.

6.12.3.6 template<typename T , typename L > void DForestBuilder< T, L >::setBoostStrapSize (size_t const *bootStrapSize*) [inline]

Set the size of bootstrap sample to be used in training.

6.12.3.7 template<typename T , typename L > void DForestBuilder< T, L >::setNumRandomFeatures (size_t const *numRandomFeatures*) [inline]

6.12.3.8 template<typename T , typename L > void DForestBuilder< T, L >::setNumTrees (size_t const *numTrees*) [inline]

Parameters

The	number of trees to train for the decision forest.

6.12.4 Member Data Documentation

6.12.4.1 template<typename T , typename L > size_t DForestBuilder< T, L >::bootStrapSize [protected]

6.12.4.2 template<typename T , typename L > DataSet<T, L> const& DForestBuilder< T, L >::ds [protected]

6.12.4.3 template<typename T , typename L > size_t DForestBuilder< T, L >::numRandomFeatures [protected]

6.12.4.4 template<typename T , typename L > size_t DForestBuilder< T, L >::numTrees [protected]

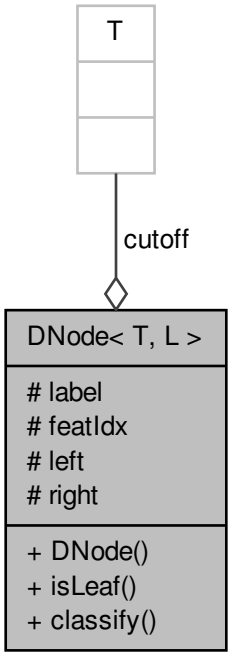
The documentation for this class was generated from the following file:

- include/dfpar.hpp

6.13 DNode< T, L > Class Template Reference

`#include <dfpar.hpp>`

Collaboration diagram for DNode< T, L >:



Public Member Functions

- DNode ()
- bool isLeaf ()
- L classify (DataSet< T, L > const &ds, size_t const idx)

Protected Attributes

- T cutoff

  The cutoff value of the branch. Tested as x >= cutoff.
- L label

  If the node is a leaf node, then the label assigned to it.
- size_t featIdx

  Feature (column) index the split is made.
- std::shared_ptr< DNode< T, L > > left

  Left child node.
- std::shared_ptr< DNode< T, L > > right

  Right child node.

Friends

- class DTreeBuilder< T, L >
- std::ostream & operator<< (std::ostream &os, DNode< T, L > &node)
- std::istream & operator>> (std::istream &is, DNode< T, L > &node)

6.13.1 Detailed Description

template<typename T, typename L>class DNode< T, L >

A single node in a decision tree.

Template Parameters

T	Typename for feature data.
L	Typename for response (label) data.

6.13.2 Constructor & Destructor Documentation

6.13.2.1 template<typename T, typename L> DNode< T, L >::DNode ( ) [inline]

6.13.3 Member Function Documentation

6.13.3.1 template<typename T, typename L> L DNode< T, L >::classify (DataSet< T, L > const & *ds*, size_t const *idx*) [inline]

Classify a row in dataset.

Parameters

ds	Dataset instance.
idx	The row index.

6.13.3.2 template<typename T, typename L> bool DNode< T, L >::isLeaf ( ) [inline]

Returns

true if node is leaf, false otherwise.

6.13.4 Friends And Related Function Documentation

6.13.4.1 template<typename T, typename L> friend class DTreeBuilder< T, L > [friend]

6.13.4.2 template<typename T, typename L> std::ostream& operator<< (std::ostream & *os,* DNode< T, L > & *node*) [friend]

6.13.4.3 template<typename T, typename L> std::istream& operator>> (std::istream & *is,* DNode< T, L > & *node*) [friend]

6.13.5 Member Data Documentation

6.13.5.1 template<typename T, typename L> T DNode< T, L >::cutoff [protected]

The cutoff value of the branch. Tested as x >= cutoff.

6.13.5.2 template<typename T, typename L> size_t DNode< T, L >::featIdx [protected]

Feature (column) index the split is made.

6.13.5.3 template<typename T, typename L> L DNode< T, L >::label [protected]

If the node is a leaf node, then the label assigned to it.

6.13.5.4 template<typename T, typename L> std::shared_ptr<DNode<T, L> > DNode< T, L >::left [protected]

Left child node.

6.13.5.5 template<typename T, typename L> std::shared_ptr<DNode<T, L> > DNode< T, L >::right [protected]

Right child node.

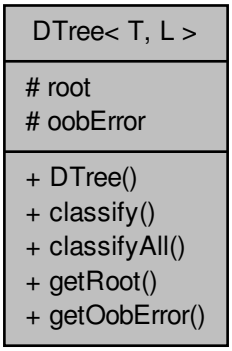
The documentation for this class was generated from the following file:

- include/dfpar.hpp

6.14 DTree< T, L > Class Template Reference

```
#include <dfpar.hpp>
```

Collaboration diagram for DTree< T, L >:



Public Member Functions

- DTree ()
- L classify (DataSet< T, L > const &ds, size_t const idx)
- void classifyAll (DataSet< T, L > const &ds, std::vector< L > &result)
- std::shared_ptr< DNode< T, L > > getRoot ()
- double getOobError () const

Protected Attributes

- std::shared_ptr< DNode< T, L > > root

  Shared pointer to the root node.
- double oobError

  oobError calculated in trainign phase.

Friends

- class DTreeBuilder< T, L >
- std::ostream & operator<< (std::ostream &os, DTree< T, L > &tree)
- std::istream & operator>> (std::istream &is, DTree< T, L > &tree)

6.14.1 Detailed Description

template<typename T, typename L>class DTree< T, L >

DTree is a component of decision forest.

Template Parameters

T	Typename for feature data.
L	Typename for response (label) data.

6.14.2 Constructor & Destructor Documentation

6.14.2.1 template<typename T, typename L> DTree< T, L >::DTree ( ) [inline]

6.14.3 Member Function Documentation

6.14.3.1 template<typename T, typename L> L DTree< T, L >::classify (DataSet< T, L > const & *ds,* size_t const *idx*) [inline]

6.14.3.2 template<typename T, typename L> void DTree< T, L >::classifyAll (DataSet< T, L > const & *ds,* std::vector< L > & *result*) [inline]

6.14.3.3 template<typename T, typename L> double DTree< T, L >::getOobError ( ) const [inline]

6.14.3.4 template<typename T, typename L> std::shared_ptr<DNode<T, L> > DTree< T, L >::getRoot ( ) [inline]

6.14.4 Friends And Related Function Documentation

6.14.4.1 template<typename T, typename L> friend class DTreeBuilder< T, L > [friend]

6.14.4.2 template<typename T, typename L> std::ostream& operator<< (std::ostream & *os,* DTree< T, L > & *tree*) [friend]

6.14.4.3 template<typename T, typename L> std::istream& operator>> (std::istream & *is,* DTree< T, L > & *tree*) [friend]

6.14.5 Member Data Documentation

6.14.5.1 template<typename T, typename L> double DTree< T, L >::oobError [protected]

oobError calculated in trainign phase.

6.14.5.2 template<typename T, typename L> std::shared_ptr<DNode<T, L> > DTree< T, L >::root [protected]

Shared pointer to the root node.

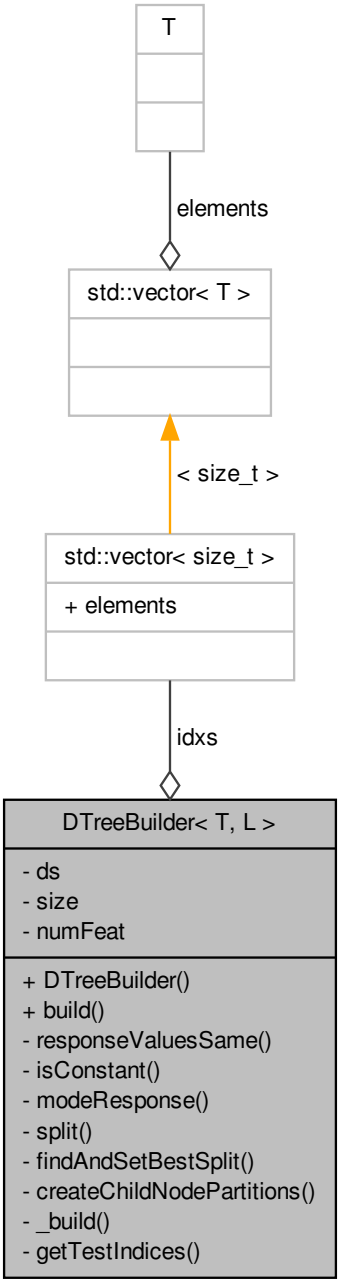
The documentation for this class was generated from the following file:

- include/dfpar.hpp

6.15 DTreeBuilder< T, L > Class Template Reference

```
#include <dfpar.hpp>
```

Collaboration diagram for DTreeBuilder< T, L >:



Public Member Functions

- DTreeBuilder (DataSet< T, L > const &ds, size_t const size, size_t const numFeat)
- void build (DTree< T, L > &tree)

Private Member Functions

- bool responseValuesSame (std::vector< size_t > const &idxs) const
- bool isConstant (std::vector< size_t > const &idxs, size_t const featIdx) const
- L modeResponse (std::vector< size_t > const &idxs) const
- std::pair< T, double > split (std::vector< size_t > const &idxs, size_t const featIdx)
- bool findAndSetBestSplit (DNode< T, L > &node, std::vector< size_t > const &idxs)
- void createChildNodePartitions (DNode< T, L > const &node, std::vector< size_t > const &idxs, std::vector< size_t > &partitionA, std::vector< size_t > &partitionB)
- std::shared_ptr< DNode< T, L > > _build (std::vector< size_t > idxs)
- std::vector< size_t > getTestIndices ()

Private Attributes

- DataSet< T, L > const & ds

  Reference to the dataset used to build the tree.
- size_t const size

  Number of rows to sample when building the tree.
- size_t const numFeat

  Number of features to sample.
- std::vector< size_t > idxs

  Sample indices for building this tree.

6.15.1 Detailed Description

template<typename T, typename L>class DTreeBuilder< T, L >

Decision Tree builder class.

Template Parameters

T	Typename for feature data.
L	Typename for response (label) data.

6.15.2 Constructor & Destructor Documentation

6.15.2.1 template<typename T, typename L> DTreeBuilder< T, L >::DTreeBuilder (DataSet< T, L > const & *ds*, size_t const *size*, size_t const *numFeat*) [inline]

Construct a new decision tree builder.

Parameters

ds	The dataset used for training.
size	The size of the bootstrap sample to use.
numFeat	The number of random features to use.

6.15.3 Member Function Documentation

6.15.3.1 template<typename T, typename L> std::shared_ptr<DNode<T, L> > DTreeBuilder< T, L >::_build (std::vector< size_t > *idxs*) [inline], [private]

6.15.3.2 template<typename T, typename L> void DTreeBuilder< T, L >::build (DTree< T, L > & *tree*) [inline]

Train a given tree instance.

6.15.3.3 template<typename T, typename L> void DTreeBuilder< T, L >::createChildNodePartitions (DNode< T, L > const & *node,* std::vector< size_t > const & *idxs,* std::vector< size_t > & *partitionA,* std::vector< size_t > & *partitionB*) [inline], [private]

Create sample indices for children of the given node.

Parameters

node	The parent node.
sampleIndices	The training indices of the parent node.
partitionA	The empty vector to store the indices of first child.
partitionB	The empty vector to store the indices of the second child.

6.15.3.4 template<typename T, typename L> bool DTreeBuilder< T, L >::findAndSetBestSplit (DNode< T, L > & *node,* std::vector< size_t > const & *idxs*) [inline], [private]

Given an empty node, choose a sample of features and splitting point for it.

Parameters

node	The empty DNode instance that is filled with information.
sampleIndices	The rows of dataset used for determining splitting point of this node.
return	true, if the initiated node is leaf node.

6.15.3.5 template<typename T, typename L> std::vector<size_t> DTreeBuilder< T, L >::getTestIndices ( ) [inline], [private]

6.15.3.6 template<typename T, typename L> bool DTreeBuilder< T, L >::isConstant (std::vector< size_t > const & *idxs,* size_t const *featIdx*) const [inline], [private]

Test if the given feature is constant in given sample indices.

6.15.3.7 template<typename T, typename L> L DTreeBuilder< T, L >::modeResponse (std::vector< size_t > const & *idxs*) const [inline], [private]

Get the most frequent response value in given sample rows.

6.15.3.8 template<typename T, typename L> bool DTreeBuilder< T, L >::responseValuesSame (std::vector< size_t > const & *idxs*) const [inline], [private]

Test if the given feature is constant in given sample indices.

6.15.3.9 template<typename T, typename L> std::pair<T, double> DTreeBuilder< T, L >::split (std::vector< size_t > const & *idxs,* size_t const *featIdx*) [inline], [private]

6.15.4 Member Data Documentation

6.15.4.1 template<typename T, typename L> DataSet<T, L> const& DTreeBuilder< T, L >::ds [private]

Reference to the dataset used to build the tree.

6.15.4.2 template<typename T, typename L> std::vector<size_t> DTreeBuilder< T, L >::idxs [private]

Sample indices for building this tree.

6.15.4.3 template<typename T, typename L> size_t const DTreeBuilder< T, L >::numFeat [private]

Number of features to sample.

6.15.4.4 template<typename T, typename L> size_t const DTreeBuilder< T, L >::size [private]

Number of rows to sample when building the tree.

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

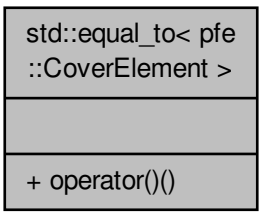
- include/dfpar.hpp

6.16 std::equal_to< pfe::CoverElement > Struct Template Reference

Specialize equal_to function for CoverElement class.

`#include <Corpus.hpp>`

Collaboration diagram for std::equal_to< pfe::CoverElement >:



Public Member Functions

- bool operator() (pfe::CoverElement const &elem1, pfe::CoverElement const &elem2) const throw ()

6.16.1 Detailed Description

template<>struct std::equal_to< pfe::CoverElement >

Specialize equal_to function for CoverElement class.

6.16.2 Member Function Documentation

6.16.2.1 bool std::equal_to< pfe::CoverElement >::operator() (pfe::CoverElement const & *elem1,* pfe::CoverElement const & *elem2*) const throw () [inline]

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

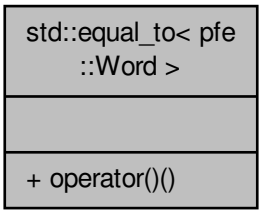
- include/Corpus.hpp

6.17 std::equal_to< pfe::Word > Struct Template Reference

Specialize equal_to function for word class.

`#include <Corpus.hpp>`

Collaboration diagram for std::equal_to< pfe::Word >:



Public Member Functions

- bool operator() (pfe::Word const &word1, pfe::Word const &word2) const throw ()

6.17.1 Detailed Description

template<>struct std::equal_to< pfe::Word >

Specialize equal_to function for word class.

6.17.2 Member Function Documentation

6.17.2.1 bool std::equal_to< pfe::Word >::operator() (pfe::Word const & *word1,* pfe::Word const & *word2*) const throw () [inline]

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

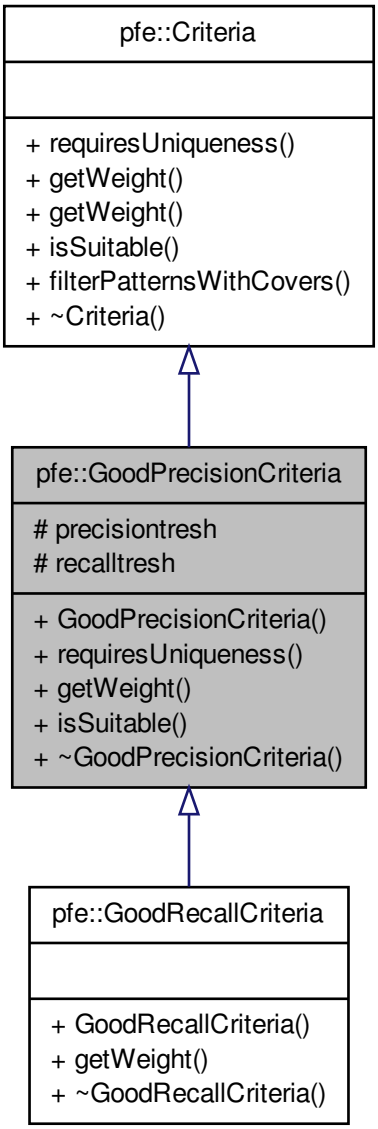
- include/Corpus.hpp

6.18 pfe::GoodPrecisionCriteria Class Reference

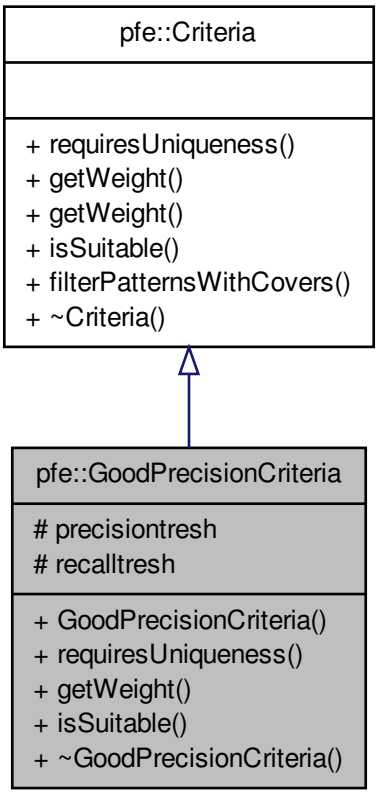
Criteria, that maximizes precision.

`#include <Pattern.hpp>`

Inheritance diagram for pfe::GoodPrecisionCriteria:



Collaboration diagram for pfe::GoodPrecisionCriteria:



Public Member Functions

- GoodPrecisionCriteria (double precisiontresh, double recalltresh)
- virtual bool requiresUniqueness () const
- virtual double getWeight (double precision, double recall)
- virtual bool isSuitable (double precision, double recall)
- virtual ~GoodPrecisionCriteria ()

Protected Attributes

- double precisiontresh
- double recalltresh

6.18.1 Detailed Description

Criteria, that maximizes precision.

6.18.2 Constructor & Destructor Documentation

6.18.2.1 pfe::GoodPrecisionCriteria::GoodPrecisionCriteria (double *precisiontresh,* double *recalltresh*)

6.18.2.2 virtual pfe::GoodPrecisionCriteria::~GoodPrecisionCriteria ( ) [inline], [virtual]

6.18.3 Member Function Documentation

6.18.3.1 double pfe::GoodPrecisionCriteria::getWeight (double *precision,* double *recall*) [virtual]

Reimplemented from pfe::Criteria.

Reimplemented in pfe::GoodRecallCriteria.

6.18.3.2 bool pfe::GoodPrecisionCriteria::isSuitable (double *precision,* double *recall*) [virtual]

Reimplemented from pfe::Criteria.

6.18.3.3 virtual bool pfe::GoodPrecisionCriteria::requiresUniqueness ( ) const [inline], [virtual]

Implements pfe::Criteria.

6.18.4 Member Data Documentation

6.18.4.1 double pfe::GoodPrecisionCriteria::precisiontresh [protected]

6.18.4.2 double pfe::GoodPrecisionCriteria::recalltresh [protected]

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

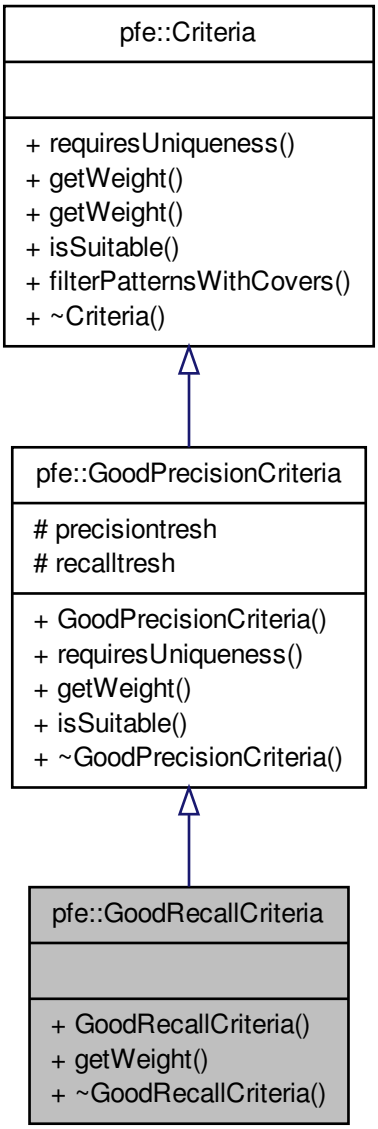
- include/Pattern.hpp
- src/Pattern.cpp

6.19 pfe::GoodRecallCriteria Class Reference

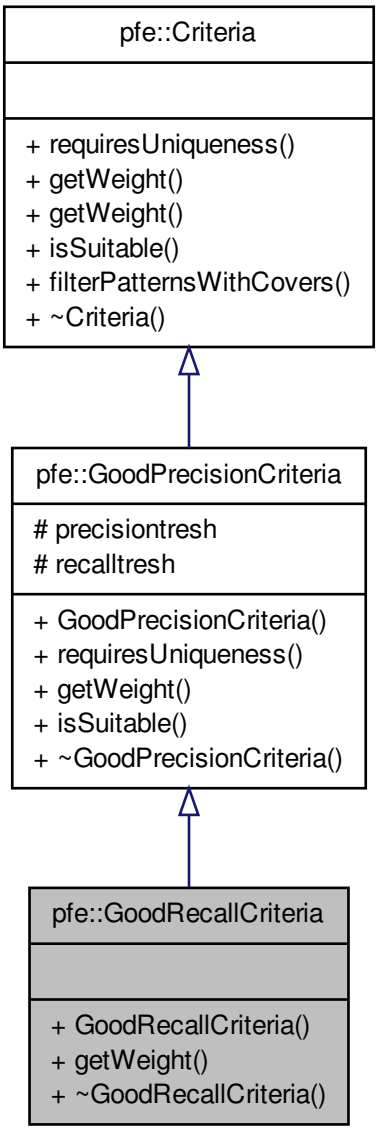
Criteria, that maximizer recall.

`#include <Pattern.hpp>`

Inheritance diagram for pfe::GoodRecallCriteria:



Collaboration diagram for pfe::GoodRecallCriteria:



Public Member Functions

- GoodRecallCriteria (double precisiontresh, double recalltresh)
- virtual double getWeight (double precision, double recall)
- virtual ~GoodRecallCriteria ()

Additional Inherited Members

6.19.1 Detailed Description

Criteria, that maximizer recall.

6.19.2 Constructor & Destructor Documentation

6.19.2.1 pfe::GoodRecallCriteria::GoodRecallCriteria (double *precisiontresh*, double *recalltresh*) [inline]

6.19.2.2 virtual pfe::GoodRecallCriteria::~GoodRecallCriteria ( ) [inline], [virtual]

6.19.3 Member Function Documentation

6.19.3.1 double pfe::GoodRecallCriteria::getWeight (double *precision*, double *recall*) [virtual]

Reimplemented from pfe::GoodPrecisionCriteria.

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

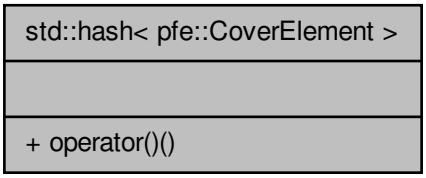
- include/Pattern.hpp
- src/Pattern.cpp

6.20 std::hash< pfe::CoverElement > Struct Template Reference

Specialize hash function for CoverElement class.

`#include <Corpus.hpp>`

Collaboration diagram for std::hash< pfe::CoverElement >:



Public Member Functions

- size_t operator() (pfe::CoverElement const &elem) const throw ()

6.20.1 Detailed Description

template<>struct std::hash< pfe::CoverElement >

Specialize hash function for CoverElement class.

6.20.2 Member Function Documentation

6.20.2.1 size_t std::hash< pfe::CoverElement >::operator() (pfe::CoverElement const & *elem*) const throw () [inline]

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

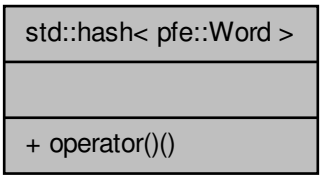
- include/Corpus.hpp

6.21 std::hash< pfe::Word > Struct Template Reference

Specialize hash function for word class.

`#include <Corpus.hpp>`

Collaboration diagram for std::hash< pfe::Word >:



Public Member Functions

- size_t operator() (pfe::Word const &word) const throw ()

6.21.1 Detailed Description

template<>struct std::hash< pfe::Word >

Specialize hash function for word class.

6.21.2 Member Function Documentation

6.21.2.1 size_t std::hash< pfe::Word >::operator() (pfe::Word const & *word*) const throw () [inline]

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

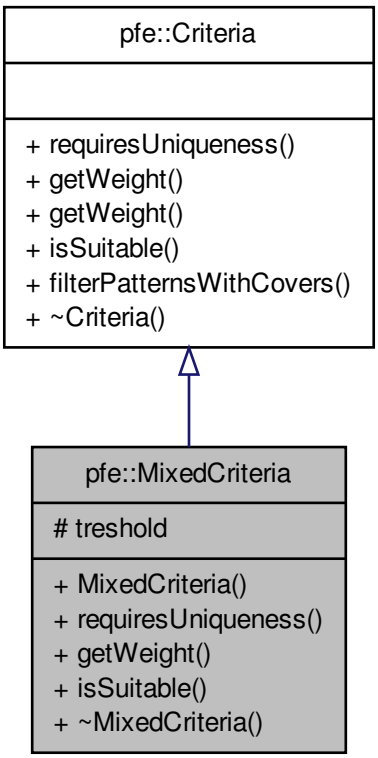
- include/Corpus.hpp

6.22 pfe::MixedCriteria Class Reference

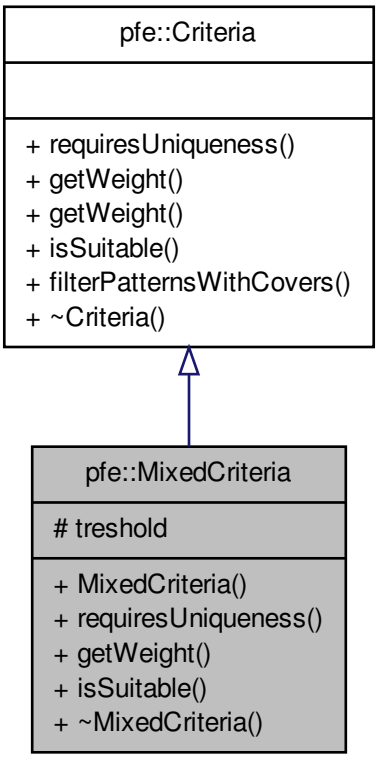
Criteria, that tries sums the precision and recall as weight.

`#include <Pattern.hpp>`

Inheritance diagram for pfe::MixedCriteria:



Collaboration diagram for pfe::MixedCriteria:



Public Member Functions

- MixedCriteria (double treshold=0.5)
- virtual bool requiresUniqueness () const
- virtual double getWeight (double precision, double recall)
- virtual bool isSuitable (double precision, double recall)
- virtual ~MixedCriteria ()

Protected Attributes

- double treshold

6.22.1 Detailed Description

Criteria, that tries sums the precision and recall as weight.

6.22.2 Constructor & Destructor Documentation

6.22.2.1 pfe::MixedCriteria::MixedCriteria (double *treshold* = 0.5)

6.22.2.2 virtual pfe::MixedCriteria::~MixedCriteria ( ) [inline], [virtual]

6.22.3 Member Function Documentation

6.22.3.1 double pfe::MixedCriteria::getWeight (double *precision,* double *recall*) [virtual]

Reimplemented from pfe::Criteria.

6.22.3.2 bool pfe::MixedCriteria::isSuitable (double *precision,* double *recall*) [virtual]

Reimplemented from pfe::Criteria.

6.22.3.3 virtual bool pfe::MixedCriteria::requiresUniqueness ( ) const [inline], [virtual]

Implements pfe::Criteria.

6.22.4 Member Data Documentation

6.22.4.1 double pfe::MixedCriteria::treshold [protected]

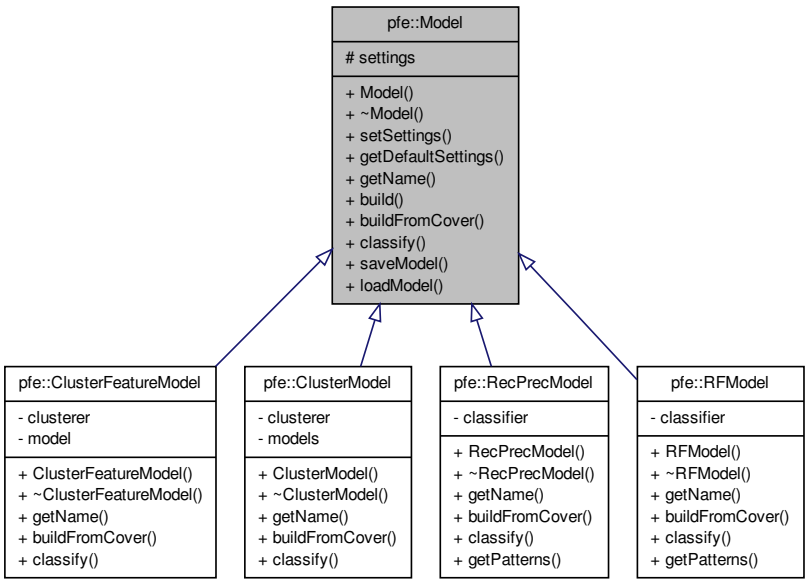
The documentation for this class was generated from the following files:

- include/Pattern.hpp
- src/Pattern.cpp

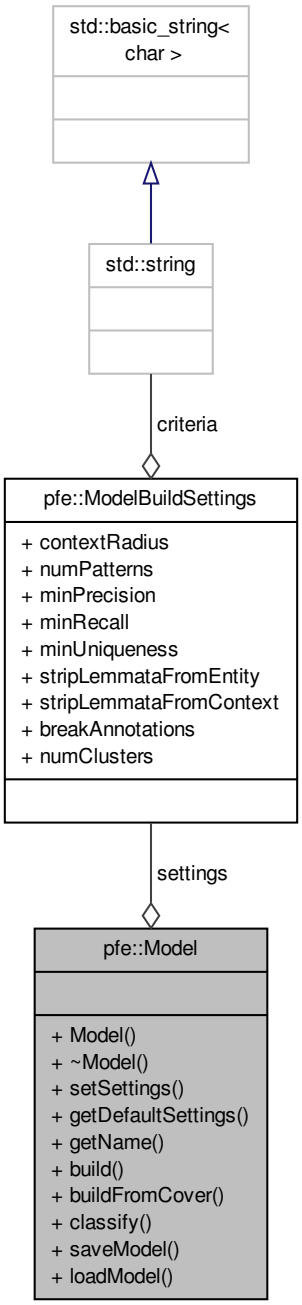
6.23 pfe::Model Class Reference

`#include <Model.hpp>`

Inheritance diagram for pfe::Model:



Collaboration diagram for pfe::Model:



Public Member Functions

- Model ()
- virtual ~Model ()
- void setSettings (ModelBuildSettings const &settings)
- virtual ModelBuildSettings getDefaultSettings ()
- virtual std::string getName () const =0

- void build (Corpus const &corpus, std::string annotString)
- virtual void buildFromCover (Corpus const &corpus, Cover const &trueCover)=0
- virtual ElementProbVector classify (Corpus const &corpus, CorpusIndex &index)=0

Static Public Member Functions

- static void saveModel (std::string path, Model const *model)
- static Model * loadModel (std::string path)

Protected Attributes

- ModelBuildSettings settings

6.23.1 Constructor & Destructor Documentation

6.23.1.1 pfe::Model::Model ( ) [inline]

6.23.1.2 virtual pfe::Model::~Model ( ) [inline], [virtual]

6.23.2 Member Function Documentation

6.23.2.1 void pfe::Model::build (Corpus const & *corpus,* std::string *annotString*)

6.23.2.2 virtual void pfe::Model::buildFromCover (Corpus const & *corpus,* Cover const & *trueCover*) [pure virtual]

Implemented in pfe::ClusterFeatureModel, pfe::ClusterModel, pfe::RecPrecModel, and pfe::RFModel.

6.23.2.3 virtual ElementProbVector pfe::Model::classify (Corpus const & *corpus,* CorpusIndex & *index*) [pure virtual]

Implemented in pfe::ClusterFeatureModel, pfe::ClusterModel, pfe::RecPrecModel, and pfe::RFModel.

6.23.2.4 ModelBuildSettings pfe::Model::getDefaultSettings ( ) [virtual]

6.23.2.5 virtual std::string pfe::Model::getName ( ) const [pure virtual]

Implemented in pfe::ClusterFeatureModel, pfe::ClusterModel, pfe::RecPrecModel, and pfe::RFModel.

6.23.2.6 Model * pfe::Model::loadModel (std::string *path*) [static]

6.23.2.7 void pfe::Model::saveModel (std::string *path,* Model const * *model*) [static]

6.23.2.8 void pfe::Model::setSettings (ModelBuildSettings const & *settings*) [inline]

6.23.3 Member Data Documentation

6.23.3.1 ModelBuildSettings pfe::Model::settings [protected]

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

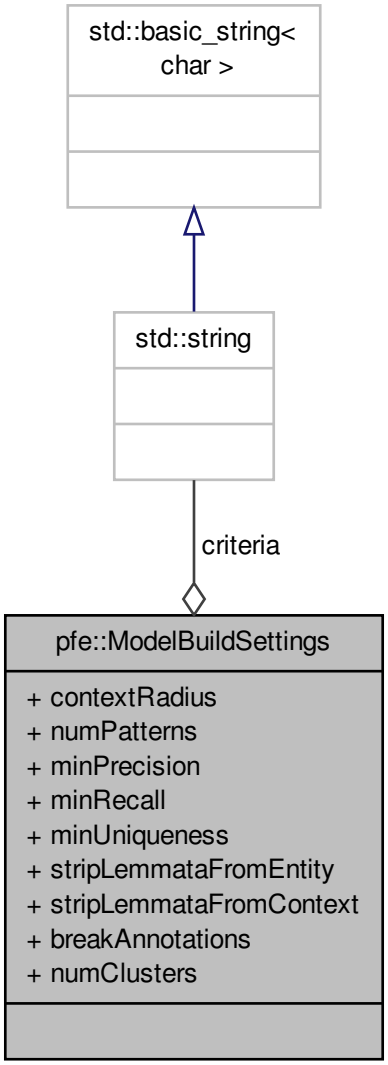
- include/Model.hpp
- src/Model.cpp

6.24 pfe::ModelBuildSettings Struct Reference

Class representing possible settings for building a model.

`#include <Model.hpp>`

Collaboration diagram for pfe::ModelBuildSettings:



Public Attributes

- size_t contextRadius
- size_t numPatterns
- double minPrecision
- double minRecall
- double minUniqueness
- bool stripLemmataFromEntity

- bool stripLemmataFromContext
- bool breakAnnotations
- std::string criteria
- size_t numClusters

Friends

- std::ostream & operator<< (std::ostream &os, ModelBuildSettings &settings)

6.24.1 Detailed Description

Class representing possible settings for building a model.

6.24.2 Friends And Related Function Documentation

6.24.2.1 std::ostream& operator<< (std::ostream & *os,* ModelBuildSettings & *settings*) [friend]

6.24.3 Member Data Documentation

6.24.3.1 bool pfe::ModelBuildSettings::breakAnnotations

6.24.3.2 size_t pfe::ModelBuildSettings::contextRadius

6.24.3.3 std::string pfe::ModelBuildSettings::criteria

6.24.3.4 double pfe::ModelBuildSettings::minPrecision

6.24.3.5 double pfe::ModelBuildSettings::minRecall

6.24.3.6 double pfe::ModelBuildSettings::minUniqueness

6.24.3.7 size_t pfe::ModelBuildSettings::numClusters

6.24.3.8 size_t pfe::ModelBuildSettings::numPatterns

6.24.3.9 bool pfe::ModelBuildSettings::stripLemmataFromContext

6.24.3.10 bool pfe::ModelBuildSettings::stripLemmataFromEntity

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

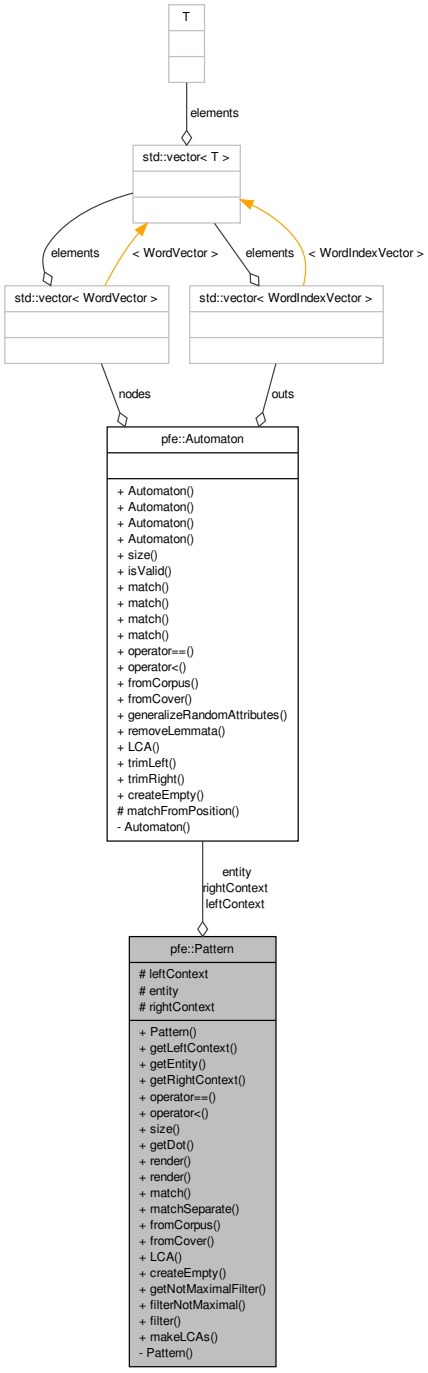
- include/Model.hpp

6.25 pfe::Pattern Class Reference

Pattern consist of left context, entity area and right context, which are all automatons.

`#include <Pattern.hpp>`

Collaboration diagram for pfe::Pattern:



Public Member Functions

- Pattern (Automaton const &leftContext, Automaton const &entity, Automaton const &rightContext)
- const Automaton & getLeftContext () const
- const Automaton & getEntity () const
- const Automaton & getRightContext () const
- bool operator== (Pattern const &other) const

- bool operator< (Pattern const &other) const
- size_t size () const
- std::string getDot (std::string const &title="Pattern") const
- std::string render (std::string const &title, const std::string &format) const
- std::string render (const char *const title, const char *const format) const
- Cover match (CorpusIndex &index) const

Static Public Member Functions

- static CoverVector matchSeparate (PatternVector const &patterns, CorpusIndex &index)
- static PatternVector fromCorpus (Corpus const &corpus, unsigned long const annotation, bool stripLemmataFromEntity=true, bool stripLemmataFromContext=true, size_t contextRadius=2)
- static PatternVector fromCover (Corpus const &corpus, Cover const &cover, bool stripLemmataFromEntity=true, bool stripLemmataFromContext=true, size_t contextRadius=2)
- static Pattern LCA (Pattern const &A, Pattern const &B)
- static Pattern createEmpty ()
- static std::vector< bool > getNotMaximalFilter (PatternVector const &patterns, CoverVector const &covers)
- static void filterNotMaximal (PatternVector &patterns, CoverVector &covers)
- static PatternVector filter (PatternVector const &patterns, DoubleVector const &values, double tresh)
- static PatternVector makeLCAs (PatternVector const &patterns)

Protected Attributes

- Automaton leftContext
- Automaton entity
- Automaton rightContext

Private Member Functions

- Pattern ()

Friends

- std::ostream & operator<< (std::ostream &os, Pattern const &pattern)
- std::istream & operator>> (std::istream &is, Pattern &pattern)

6.25.1 Detailed Description

Pattern consist of left context, entity area and right context, which are all automatons.

6.25.2 Constructor & Destructor Documentation

6.25.2.1 pfe::Pattern::Pattern ( ) [private]

6.25.2.2 pfe::Pattern::Pattern (Automaton const & *leftContext*, Automaton const & *entity*, Automaton const & *rightContext*)

6.25.3 Member Function Documentation

6.25.3.1 Pattern pfe::Pattern::createEmpty ( ) [static]

6.25.3.2 **PatternVector pfe::Pattern::filter (PatternVector const & *patterns,* DoubleVector const & *values,* double *tresh*) [static]**

Filter patterns, whose metric is greater or equal to treshold. If treshold is negative, the treshold is still used as a positive value, but instead we keep patterns with metric less or equal to abs(tresh).

6.25.3.3 **void pfe::Pattern::filterNotMaximal (PatternVector & *patterns,* CoverVector & *covers*) [static]**

6.25.3.4 **PatternVector pfe::Pattern::fromCorpus (Corpus const & *corpus,* unsigned long const *annotation,* bool *stripLemmataFromEntity* = true, bool *stripLemmataFromContext* = true, size_t *contextRadius* = 2) [static]**

6.25.3.5 **PatternVector pfe::Pattern::fromCover (Corpus const & *corpus,* Cover const & *cover,* bool *stripLemmataFromEntity* = true, bool *stripLemmataFromContext* = true, size_t *contextRadius* = 2) [static]**

6.25.3.6 **std::string pfe::Pattern::getDot (std::string const & *title* = "Pattern") const**

6.25.3.7 **const Automaton& pfe::Pattern::getEntity ( ) const [inline]**

6.25.3.8 **const Automaton& pfe::Pattern::getLeftContext ( ) const [inline]**

6.25.3.9 **std::vector< bool > pfe::Pattern::getNotMaximalFilter (PatternVector const & *patterns,* CoverVector const & *covers*) [static]**

6.25.3.10 **const Automaton& pfe::Pattern::getRightContext ( ) const [inline]**

6.25.3.11 **Pattern pfe::Pattern::LCA (Pattern const & *A,* Pattern const & *B*) [static]**

6.25.3.12 **PatternVector pfe::Pattern::makeLCAs (PatternVector const & *patterns*) [static]**

6.25.3.13 **Cover pfe::Pattern::match (CorpusIndex & *index*) const**

6.25.3.14 **CoverVector pfe::Pattern::matchSeparate (PatternVector const & *patterns,* CorpusIndex & *index*) [static]**

6.25.3.15 **bool pfe::Pattern::operator< (Pattern const & *other*) const**

6.25.3.16 **bool pfe::Pattern::operator== (Pattern const & *other*) const**

6.25.3.17 **std::string pfe::Pattern::render (std::string const & *title,* const std::string & *format*) const**

6.25.3.18 **std::string pfe::Pattern::render (const char *const *title,* const char *const *format*) const**

6.25.3.19 **size_t pfe::Pattern::size ( ) const [inline]**

6.25.4 Friends And Related Function Documentation

6.25.4.1 **std::ostream& operator<< (std::ostream & *os,* Pattern const & *pattern*) [friend]**

6.25.4.2 **std::istream& operator>> (std::istream & *is,* Pattern & *pattern*) [friend]**

6.25.5 Member Data Documentation

6.25.5.1 **Automaton pfe::Pattern::entity [protected]**

6.25.5.2 Automaton pfe::Pattern::leftContext [protected]

6.25.5.3 Automaton pfe::Pattern::rightContext [protected]

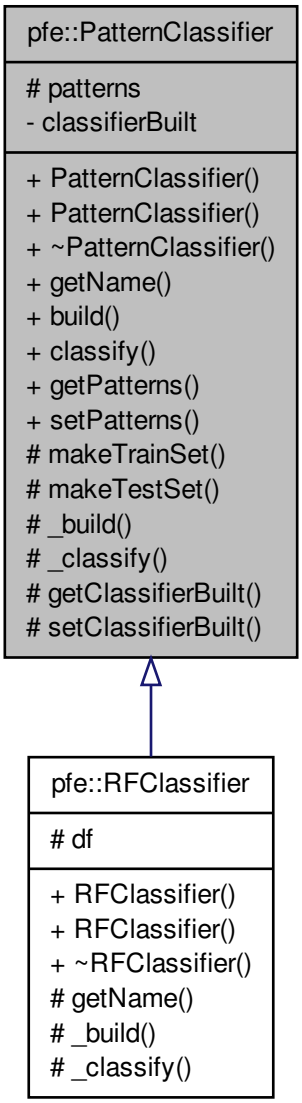
The documentation for this class was generated from the following files:

- include/Pattern.hpp
- src/Pattern.cpp

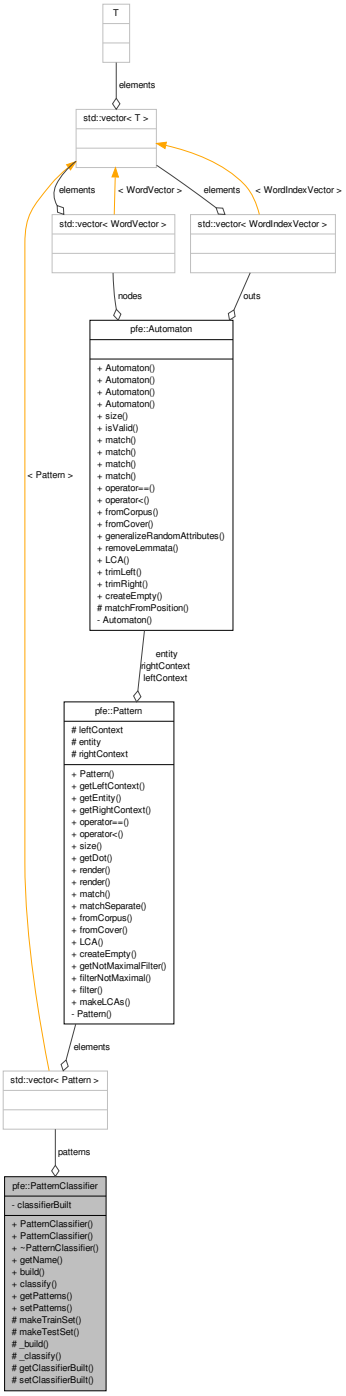
6.26 pfe::PatternClassifier Class Reference

`#include <Classifier.hpp>`

Inheritance diagram for pfe::PatternClassifier:



Collaboration diagram for pfe::PatternClassifier:



Public Member Functions

- PatternClassifier ()
- PatternClassifier (PatternVector const &patterns)
- virtual ~PatternClassifier ()
- virtual std::string getName () const
- void build (Corpus const &corpus, CoverVector const &covers, Cover const &trueCover)

- ElementProbVector classify (Corpus const &corpus, CorpusIndex &index)
- PatternVector const & getPatterns () const
- void setPatterns (PatternVector const &patterns)

Protected Member Functions

- dfpar::DataSet< int, int > makeTrainSet (Corpus const &corpus, CoverVector const &covers, Cover const &trueCover)
- dfpar::DataSet< int, int > makeTestSet (Corpus const &corpus, Cover const &all, CoverVector const &covers)
- virtual void _build (dfpar::DataSet< int, int > const &trainSet)=0
- virtual DoubleVector _classify (dfpar::DataSet< int, int > const &testSet)=0
- bool getClassifierBuilt () const
- void setClassifierBuilt (bool c)

Protected Attributes

- PatternVector patterns

Private Attributes

- bool classifierBuilt

6.26.1 Constructor & Destructor Documentation

6.26.1.1 pfe::PatternClassifier::PatternClassifier ( )

When using this constructor, caller must ensure that there will be enough patterns, when building it.

6.26.1.2 pfe::PatternClassifier::PatternClassifier (PatternVector const & *patterns*)

6.26.1.3 virtual pfe::PatternClassifier::~PatternClassifier ( ) [inline], [virtual]

6.26.2 Member Function Documentation

6.26.2.1 virtual void pfe::PatternClassifier::_build (dfpar::DataSet< int, int > const & *trainSet*) [protected], [pure virtual]

Implemented in pfe::RFClassifier.

6.26.2.2 virtual DoubleVector pfe::PatternClassifier::_classify (dfpar::DataSet< int, int > const & *testSet*) [protected], [pure virtual]

Implemented in pfe::RFClassifier.

6.26.2.3 void pfe::PatternClassifier::build (Corpus const & *corpus,* CoverVector const & *covers,* Cover const & *trueCover*)

6.26.2.4 ElementProbVector pfe::PatternClassifier::classify (Corpus const & *corpus,* CorpusIndex & *index*)

6.26.2.5 bool pfe::PatternClassifier::getClassifierBuilt ( ) const [inline], [protected]

6.26.2.6 virtual std::string pfe::PatternClassifier::getName ( ) const [inline], [virtual]

Reimplemented in pfe::RFClassifier.

6.26.2.7 PatternVector const& pfe::PatternClassifier::getPatterns ( ) const [inline]

6.26.2.8 dfpar::DataSet< int, int > pfe::PatternClassifier::makeTestSet (Corpus const & *corpus,* Cover const & *all,* CoverVector const & *covers*) [protected]

6.26.2.9 dfpar::DataSet< int, int > pfe::PatternClassifier::makeTrainSet (Corpus const & *corpus,* CoverVector const & *covers,* Cover const & *trueCover*) [protected]

6.26.2.10 void pfe::PatternClassifier::setClassifierBuilt (bool *c*) [inline], [protected]

6.26.2.11 void pfe::PatternClassifier::setPatterns (PatternVector const & *patterns*) [inline]

6.26.3 Member Data Documentation

6.26.3.1 bool pfe::PatternClassifier::classifierBuilt [private]

6.26.3.2 PatternVector pfe::PatternClassifier::patterns [protected]

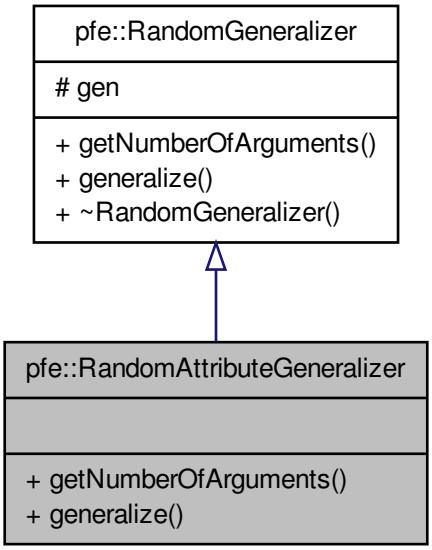
The documentation for this class was generated from the following files:

- include/Classifier.hpp
- src/Classifier.cpp

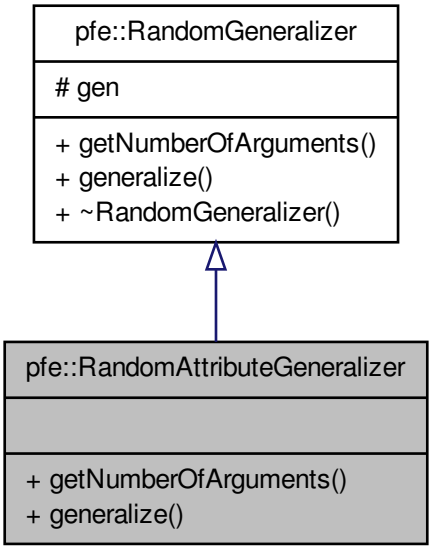
6.27 pfe::RandomAttributeGeneralizer Class Reference

```
#include <Pattern.hpp>
```

Inheritance diagram for pfe::RandomAttributeGeneralizer:



Collaboration diagram for pfe::RandomAttributeGeneralizer:



Public Member Functions

- virtual size_t getNumberOfArguments () const

  How many source patterns does this generalizer need.

- virtual Pattern generalize (PatternVector const &arguments)

  Given the arguments, create a child vector.

Additional Inherited Members

6.27.1 Member Function Documentation

6.27.1.1 Pattern pfe::RandomAttributeGeneralizer::generalize (PatternVector const & *arguments*) [virtual]

Given the arguments, create a child vector.

Implements pfe::RandomGeneralizer.

6.27.1.2 size_t pfe::RandomAttributeGeneralizer::getNumberOfArguments ( ) const [virtual]

How many source patterns does this generalizer need.

Implements pfe::RandomGeneralizer.

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

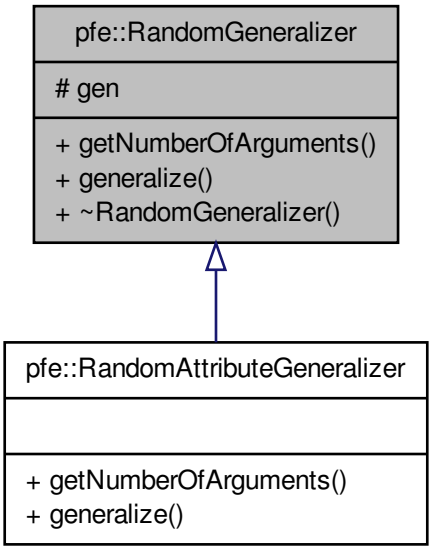
- include/Pattern.hpp
- src/Pattern.cpp

6.28 pfe::RandomGeneralizer Class Reference

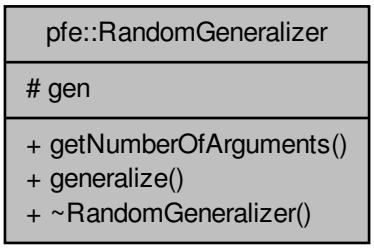
Base class for random generalizers for optimisation tasks.

`#include <Pattern.hpp>`

Inheritance diagram for pfe::RandomGeneralizer:



Collaboration diagram for pfe::RandomGeneralizer:



Public Member Functions

- virtual size_t getNumberOfArguments () const =0

  How many source patterns does this generalizer need.
- virtual Pattern generalize (PatternVector const &arguments)=0

  Given the arguments, create a child vector.
- virtual ~RandomGeneralizer ()

Static Protected Attributes

- static boost::random::mt19937 gen

6.28.1 Detailed Description

Base class for random generalizers for optimisation tasks.

6.28.2 Constructor & Destructor Documentation

6.28.2.1 virtual pfe::RandomGeneralizer::~RandomGeneralizer ( ) [inline], [virtual]

6.28.3 Member Function Documentation

6.28.3.1 virtual Pattern pfe::RandomGeneralizer::generalize (PatternVector const & *arguments*) [pure virtual]

Given the arguments, create a child vector.

Implemented in pfe::RandomAttributeGeneralizer.

6.28.3.2 virtual size_t pfe::RandomGeneralizer::getNumberOfArguments ( ) const [pure virtual]

How many source patterns does this generalizer need.

Implemented in pfe::RandomAttributeGeneralizer.

6.28.4 Member Data Documentation

6.28.4.1 boost::random::mt19937 pfe::RandomGeneralizer::gen [static], [protected]

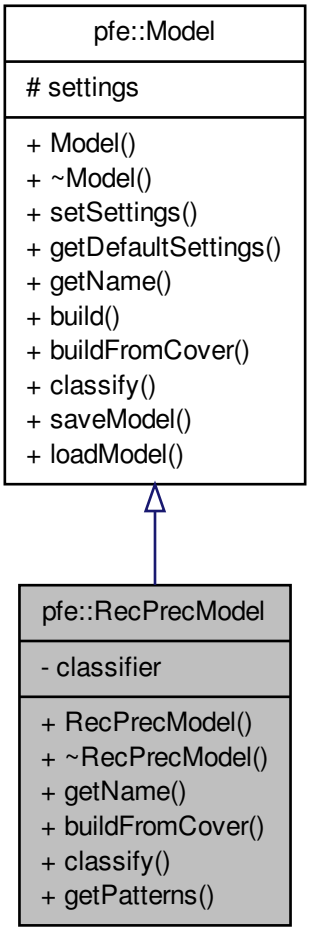
The documentation for this class was generated from the following files:

- include/Pattern.hpp
- src/Pattern.cpp

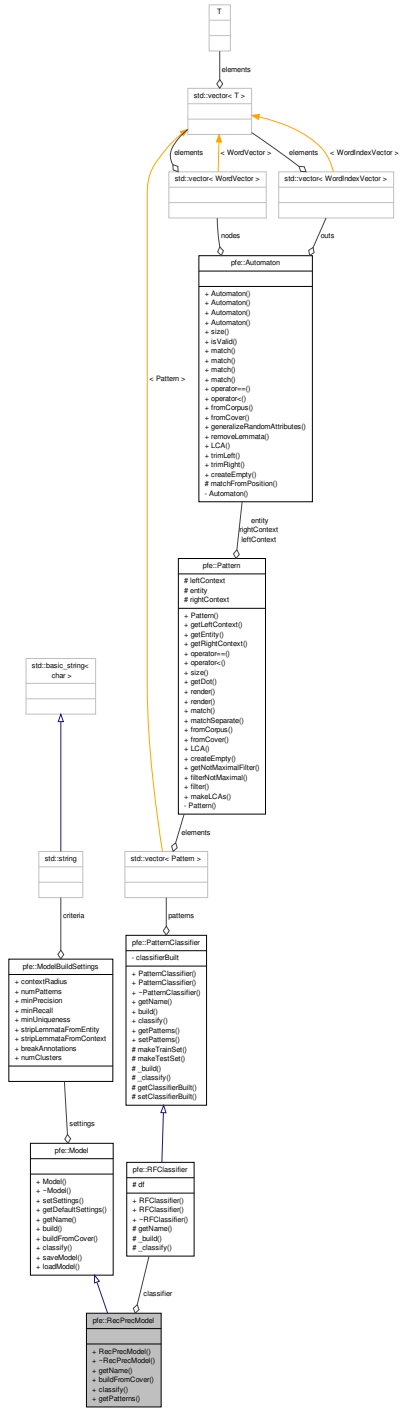
6.29 pfe::RecPrecModel Class Reference

```
#include <Model.hpp>
```

Inheritance diagram for pfe::RecPrecModel:



Collaboration diagram for pfe::RecPrecModel:



Public Member Functions

- RecPrecModel ()
- virtual ~RecPrecModel ()
- virtual std::string getName () const
- virtual void buildFromCover (Corpus const &corpus, Cover const &trueCover)
- virtual ElementProbVector classify (Corpus const &corpus, CorpusIndex &index)
- PatternVector getPatterns () const

Private Attributes

- RFClassifier classifier

Friends

- std::ostream & operator<< (std::ostream &os, RecPrecModel &model)
- std::istream & operator>> (std::istream &is, RecPrecModel &model)

Additional Inherited Members

6.29.1 Detailed Description

RecPrecModel. trains a combo classifier, where first set of patterns try to achieve maximal recall end second part of the patterns try to eliminate false positives as accurately as possible.

6.29.2 Constructor & Destructor Documentation

6.29.2.1 pfe::RecPrecModel::RecPrecModel ( ) [inline]

6.29.2.2 virtual pfe::RecPrecModel::~RecPrecModel ( ) [inline], [virtual]

6.29.3 Member Function Documentation

6.29.3.1 void pfe::RecPrecModel::buildFromCover (Corpus const & *corpus,* Cover const & *trueCover*) [virtual]

Implements pfe::Model.

6.29.3.2 ElementProbVector pfe::RecPrecModel::classify (Corpus const & *corpus,* CorpusIndex & *index*) [virtual]

Implements pfe::Model.

6.29.3.3 virtual std::string pfe::RecPrecModel::getName ( ) const [inline], [virtual]

Implements pfe::Model.

6.29.3.4 PatternVector pfe::RecPrecModel::getPatterns ( ) const [inline]

6.29.4 Friends And Related Function Documentation

6.29.4.1 std::ostream& operator<< (std::ostream & *os,* RecPrecModel & *model*) [friend]

6.29.4.2 std::istream& operator>> (std::istream & *is,* RecPrecModel & *model*) [friend]

6.29.5 Member Data Documentation

6.29.5.1 RFClassifier pfe::RecPrecModel::classifier [private]

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

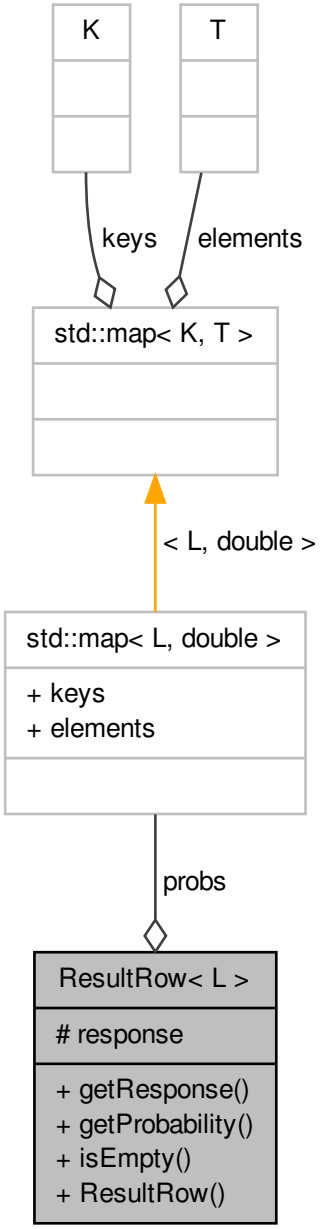
- include/Model.hpp

- src/Model.cpp

6.30 ResultRow< L > Struct Template Reference

`#include <dfpar.hpp>`

Collaboration diagram for ResultRow< L >:



Public Member Functions

- L getResponse () const
- double getProbability (L const &y)
- bool isEmpty () const
- ResultRow (L response, std::map< L, double > const &probs)

Protected Attributes

- L response

  Response value.
- std::map< L, double > probs

  Map of response values and their probabilities.

6.30.1 Detailed Description

template<typename L>struct ResultRow< L >

ResultRow describes the preferred response value (label) with probabilities of all candidates considered for classification.

Template Parameters

L	Typename for response (label) data.

6.30.2 Constructor & Destructor Documentation

6.30.2.1 template<typename L> ResultRow< L >::ResultRow (L *response*, std::map< L, double > const & *probs*) [inline]

Construct a Resultrow instance from given data

6.30.3 Member Function Documentation

6.30.3.1 template<typename L> double ResultRow< L >::getProbability (L const & *y*) [inline]

Return the probability for response value (label) y.

Parameters

y	The response value (label).

Returns

the probability for label y.

6.30.3.2 template<typename L> L ResultRow< L >::getResponse ( ) const [inline]

Get the copy of response (label) of this row.

6.30.3.3 template<typename L> bool ResultRow< L >::isEmpty ( ) const [inline]

Is the resultrow initialized.

Returns

true or false.

6.30.4 Member Data Documentation

6.30.4.1 template<typename L> std::map<L, double> ResultRow< L >::probs [protected]

Map of response values and their probabilities.

6.30.4.2 template<typename L> L ResultRow< L >::response [protected]

Response value.

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

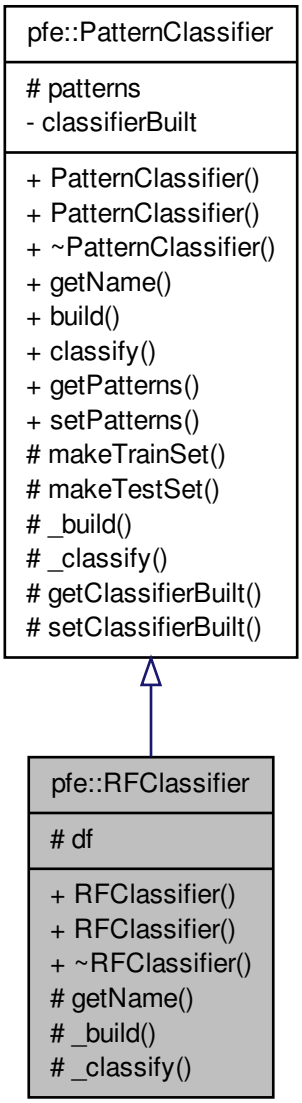
- include/dfpar.hpp

6.31 pfe::RFClassifier Class Reference

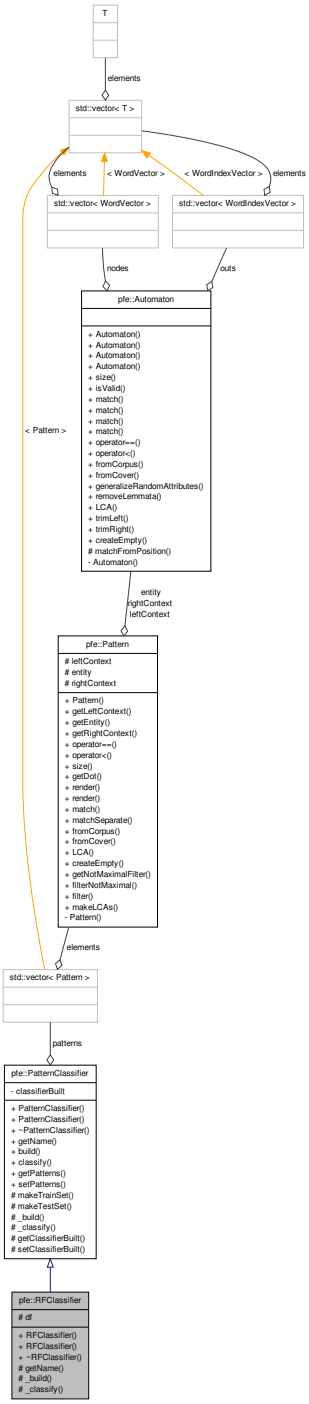
Random forest classifier.

`#include <Classifier.hpp>`

Inheritance diagram for pfe::RFClassifier:



Collaboration diagram for pfe::RFClassifier:



Public Member Functions

- RFClassifier ()
- RFClassifier (PatternVector const &patterns)
- virtual ~RFClassifier ()

Protected Member Functions

- virtual std::string getName () const
- virtual void _build (dfpar::DataSet< int, int > const &trainSet)
- virtual DoubleVector _classify (dfpar::DataSet< int, int > const &testSet)

Protected Attributes

- dfpar::DForest< int, int > df

Friends

- std::ostream & operator<< (std::ostream &os, RFClassifier &classifier)
- std::istream & operator>> (std::istream &is, RFClassifier &classifier)

6.31.1 Detailed Description

Random forest classifier.

6.31.2 Constructor & Destructor Documentation

6.31.2.1 pfe::RFClassifier::RFClassifier ( ) [inline]

6.31.2.2 pfe::RFClassifier::RFClassifier (PatternVector const & *patterns*) [inline]

6.31.2.3 pfe::RFClassifier::~RFClassifier ( ) [virtual]

6.31.3 Member Function Documentation

6.31.3.1 void pfe::RFClassifier::_build (dfpar::DataSet< int, int > const & *trainSet*) [protected], [virtual]

Implements pfe::PatternClassifier.

6.31.3.2 DoubleVector pfe::RFClassifier::_classify (dfpar::DataSet< int, int > const & *testSet*) [protected], [virtual]

Implements pfe::PatternClassifier.

6.31.3.3 virtual std::string pfe::RFClassifier::getName ( ) const [inline], [protected], [virtual]

Reimplemented from pfe::PatternClassifier.

6.31.4 Friends And Related Function Documentation

6.31.4.1 std::ostream& operator<< (std::ostream & *os,* RFClassifier & *classifier*) [friend]

6.31.4.2 std::istream& operator>> (std::istream & *is,* RFClassifier & *classifier*) [friend]

6.31.5 Member Data Documentation

6.31.5.1 dfpar::DForest<int, int> pfe::RFClassifier::df [protected]

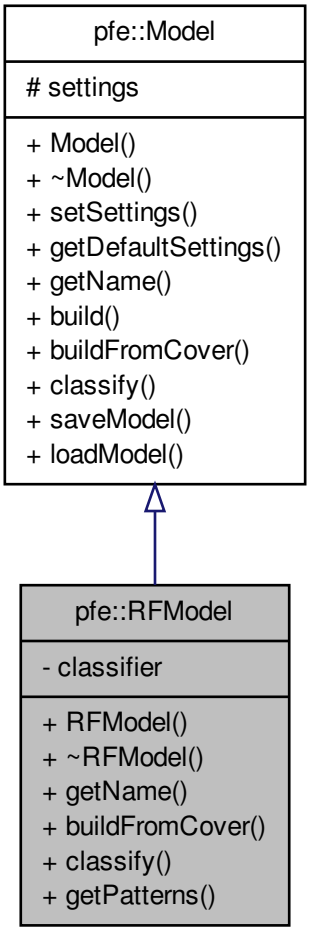
The documentation for this class was generated from the following files:

- include/Classifier.hpp
- src/Classifier.cpp

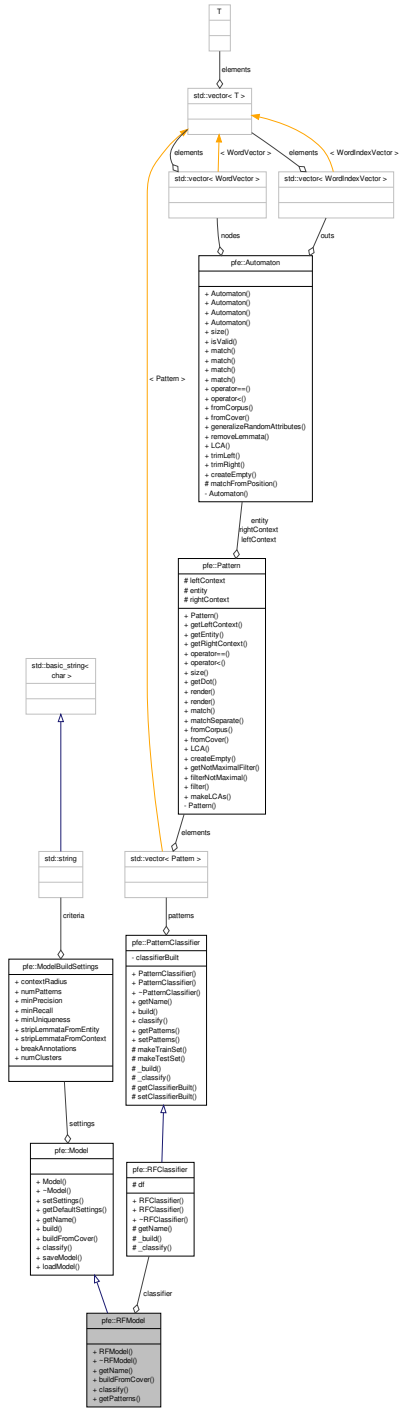
6.32 pfe::RFModel Class Reference

`#include <Model.hpp>`

Inheritance diagram for pfe::RFModel:



Collaboration diagram for pfe::RFModel:



Public Member Functions

- RFModel ()
- virtual ~RFModel ()
- virtual std::string getName () const
- virtual void buildFromCover (Corpus const &corpus, Cover const &trueCover)
- virtual ElementProbVector classify (Corpus const &corpus, CorpusIndex &index)
- PatternVector getPatterns () const

Private Attributes

- RFClassifier classifier

Friends

- std::ostream & operator<< (std::ostream &os, RFModel &model)
- std::istream & operator>> (std::istream &is, RFModel &model)

Additional Inherited Members

6.32.1 Constructor & Destructor Documentation

6.32.1.1 pfe::RFModel::RFModel ( )

6.32.1.2 virtual pfe::RFModel::~RFModel ( ) [inline], [virtual]

6.32.2 Member Function Documentation

6.32.2.1 void pfe::RFModel::buildFromCover (Corpus const & *corpus,* Cover const & *trueCover*) [virtual]

Implements pfe::Model.

6.32.2.2 ElementProbVector pfe::RFModel::classify (Corpus const & *corpus,* CorpusIndex & *index*) [virtual]

Implements pfe::Model.

6.32.2.3 virtual std::string pfe::RFModel::getName ( ) const [inline], [virtual]

Implements pfe::Model.

6.32.2.4 PatternVector pfe::RFModel::getPatterns ( ) const [inline]

6.32.3 Friends And Related Function Documentation

6.32.3.1 std::ostream& operator<< (std::ostream & *os,* RFModel & *model*) [friend]

6.32.3.2 std::istream& operator>> (std::istream & *is,* RFModel & *model*) [friend]

6.32.4 Member Data Documentation

6.32.4.1 RFClassifier pfe::RFModel::classifier [private]

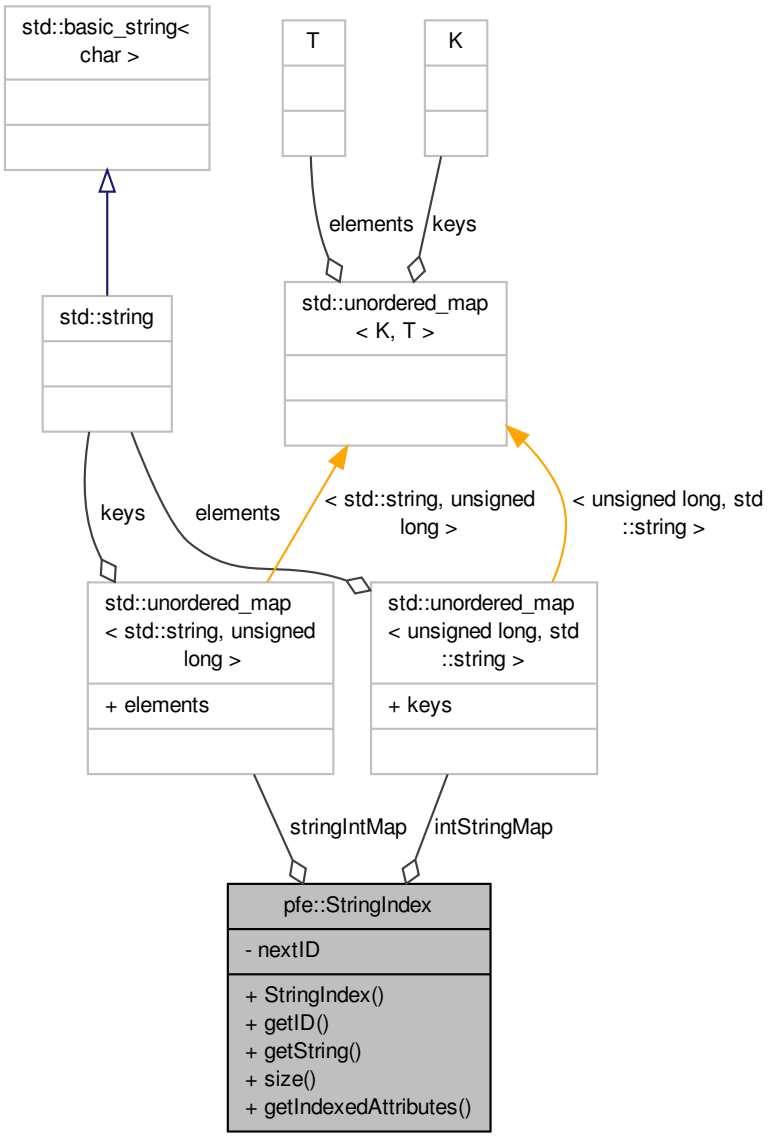
The documentation for this class was generated from the following files:

- include/Model.hpp
- src/Model.cpp

6.33 pfe::StringIndex Class Reference

`#include <StringIndex.hpp>`

Collaboration diagram for pfe::StringIndex:



Public Member Functions

- StringIndex ()
- unsigned long getID (std::string const &s)

  Given an string, return an unique ID for it.
- std::string getString (unsigned long const ID)

  Given and unique ID, return a string.
- size_t size () const
- std::vector< unsigned long > getIndexedAttributes ()

Private Attributes

- unsigned long nextID
- std::unordered_map < std::string, unsigned long > stringIntMap
- std::unordered_map< unsigned long, std::string > intStringMap

6.33.1 Detailed Description

StringIndex class is used for mapping strings to integers.

The mapping makes possible to use unique integer values instead of strings in code that only requires comparing strings.

6.33.2 Constructor & Destructor Documentation

6.33.2.1 pfe::StringIndex::StringIndex ( )

6.33.3 Member Function Documentation

6.33.3.1 unsigned long pfe::StringIndex::getID (std::string const & *s*)

Given an string, return an unique ID for it.

6.33.3.2 std::vector< unsigned long > pfe::StringIndex::getIndexedAttributes ( )

6.33.3.3 std::string pfe::StringIndex::getString (unsigned long const *ID*)

Given and unique ID, return a string.

6.33.3.4 size_t pfe::StringIndex::size ( ) const [inline]

6.33.4 Member Data Documentation

6.33.4.1 std::unordered_map<unsigned long, std::string> pfe::StringIndex::intStringMap [private]

6.33.4.2 unsigned long pfe::StringIndex::nextID [private]

6.33.4.3 std::unordered_map<std::string, unsigned long> pfe::StringIndex::stringIntMap [private]

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

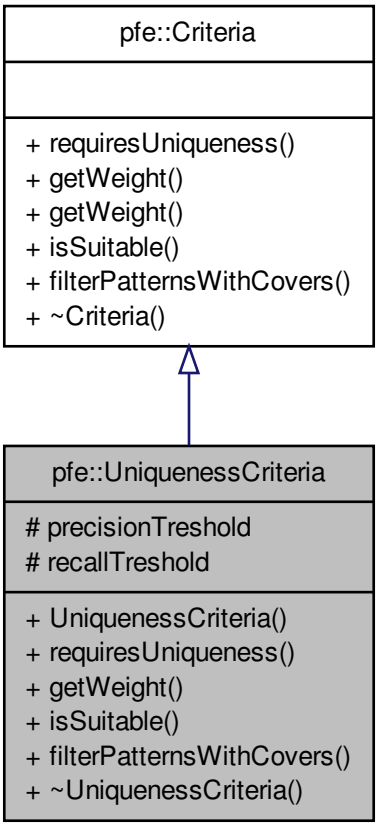
- include/StringIndex.hpp
- src/StringIndex.cpp

6.34 pfe::UniquenessCriteria Class Reference

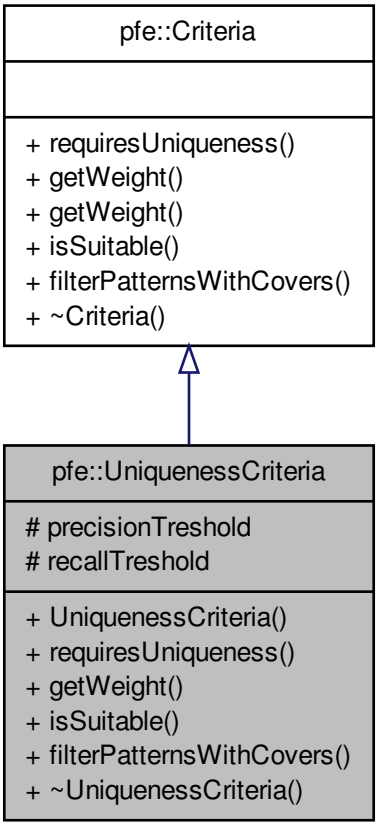
Criteria, that tries to maximize uniqueness.

```
#include <Pattern.hpp>
```

Inheritance diagram for pfe::UniquenessCriteria:



Collaboration diagram for pfe::UniquenessCriteria:



Public Member Functions

- UniquenessCriteria (double precisionTreshold=0.5, double recallTreshold=0.05)
- virtual bool requiresUniqueness () const
- virtual double getWeight (double precision, double recall, double uniqueness)
- virtual bool isSuitable (double precision, double recall)
- virtual DoubleVector filterPatternsWithCovers (PatternVector &patterns, CoverVector &covers, Cover const &trueCover)
- virtual ~UniquenessCriteria ()

Protected Attributes

- double precisionTreshold
- double recallTreshold

6.34.1 Detailed Description

Criteria, that tries to maximize uniqueness.

6.34.2 Constructor & Destructor Documentation

6.34.2.1 pfe::UniquenessCriteria::UniquenessCriteria (double *precisionTreshold* = 0.5, double *recallTreshold* = 0.05)

6.34.2.2 virtual pfe::UniquenessCriteria::~UniquenessCriteria ( ) [inline], [virtual]

6.34.3 Member Function Documentation

6.34.3.1 DoubleVector pfe::UniquenessCriteria::filterPatternsWithCovers (PatternVector & *patterns*, CoverVector & *covers*, Cover const & *trueCover*) [virtual]

Reimplemented from pfe::Criteria.

6.34.3.2 double pfe::UniquenessCriteria::getWeight (double *precision*, double *recall*, double *uniqueness*) [virtual]

Reimplemented from pfe::Criteria.

6.34.3.3 bool pfe::UniquenessCriteria::isSuitable (double *precision*, double *recall*) [virtual]

Reimplemented from pfe::Criteria.

6.34.3.4 virtual bool pfe::UniquenessCriteria::requiresUniqueness ( ) const [inline], [virtual]

Implements pfe::Criteria.

6.34.4 Member Data Documentation

6.34.4.1 double pfe::UniquenessCriteria::precisionTreshold [protected]

6.34.4.2 double pfe::UniquenessCriteria::recallTreshold [protected]

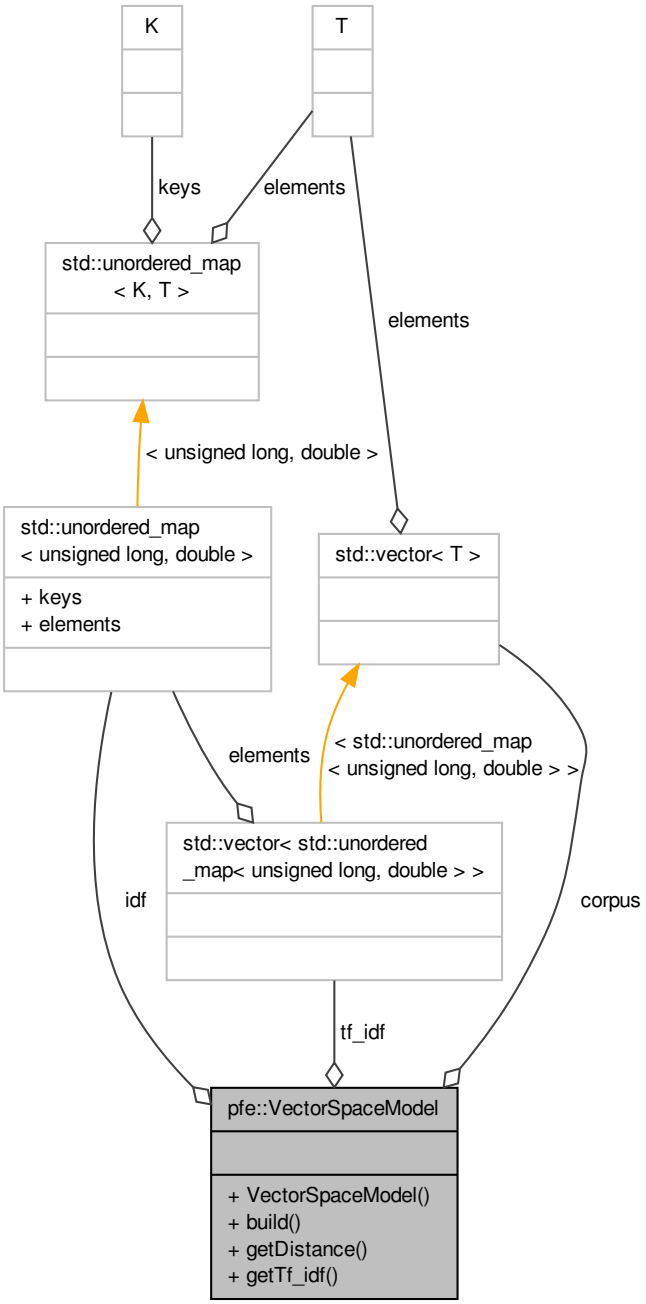
The documentation for this class was generated from the following files:

- include/Pattern.hpp
- src/Pattern.cpp

6.35 pfe::VectorSpaceModel Class Reference

```
#include <Clustering.hpp>
```

Collaboration diagram for pfe::VectorSpaceModel:



Public Member Functions

- VectorSpaceModel (Corpus const &corpus)
- void build ()

  Function that constructs the vector space model.
- double getDistance (size_t idx1, size_t idx2)

  Get the distance of two sentence determined by given indices.

- std::vector < std::unordered_map< unsigned long, double > > getTf_idf ()

Protected Attributes

- Corpus const & corpus

  Reference to the corpus we are working with.
- std::unordered_map< unsigned long, double > idf

  Idf weights for words.
- std::vector < std::unordered_map< unsigned long, double > > tf_idf

  Tf-idf weights for words in each corpus document;.

Friends

- class Clusterer

6.35.1 Detailed Description

Class implementing VectorSpaceModel for calculating the similarities of different documents of a corpus.

6.35.2 Constructor & Destructor Documentation

6.35.2.1 pfe::VectorSpaceModel::VectorSpaceModel (Corpus const & *corpus*) [inline]

6.35.3 Member Function Documentation

6.35.3.1 void pfe::VectorSpaceModel::build ( )

Function that constructs the vector space model.

6.35.3.2 double pfe::VectorSpaceModel::getDistance (size_t *idx1*, size_t *idx2*)

Get the distance of two sentence determined by given indices.

6.35.3.3 std::vector< std::unordered_map< unsigned long, double > > pfe::VectorSpaceModel::getTf_idf ( )

6.35.4 Friends And Related Function Documentation

6.35.4.1 friend class Clusterer [friend]

6.35.5 Member Data Documentation

6.35.5.1 Corpus const& pfe::VectorSpaceModel::corpus [protected]

Reference to the corpus we are working with.

6.35.5.2 std::unordered_map<unsigned long, double> pfe::VectorSpaceModel::idf [protected]

Idf weights for words.

6.35.5.3 std::vector<std::unordered_map<unsigned long, double> > pfe::VectorSpaceModel::tf_idf [protected]

Tf-idf weights for words in each corpus document;.

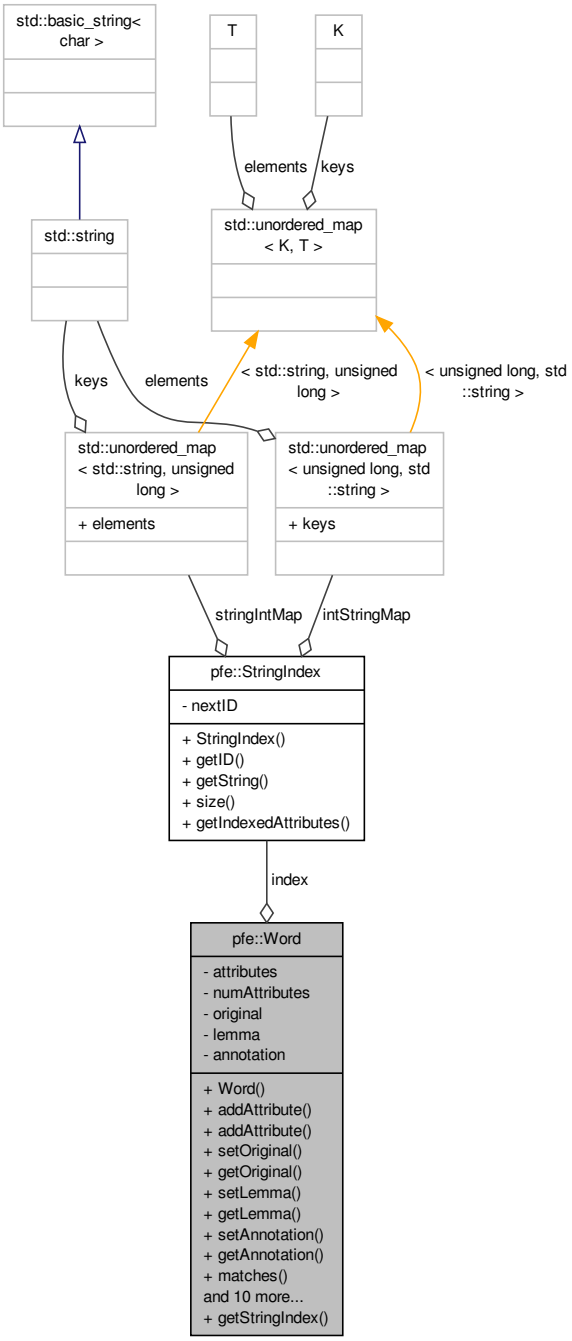
The documentation for this class was generated from the following files:

- include/Clustering.hpp
- src/Clustering.cpp

6.36 pfe::Word Class Reference

```
#include <Corpus.hpp>
```

Collaboration diagram for pfe::Word:



Public Member Functions

- Word ()

  Default constructor. Creates a default empty word.
- void addAttribute (std::string const &attribute)
- void addAttribute (unsigned long const attribute, bool toEnd=false)
- void setOriginal (std::string const &original)

Set the string that represents the original representation of the word.

- unsigned long getOriginal () const

  Get the unique integer that represents the original string of the word.
- void setLemma (std::string const &lemma)

  Set the lemma of the word.
- unsigned long getLemma () const

  Get the lemma ID.
- void setAnnotation (std::string const &annotation)

  Set the string that declares the annotation for the word.
- unsigned long getAnnotation () const

  Get the unique integer that declares the annotation for the word.
- bool matches (Word const &other) const
- Word LCA (Word const &other) const
- bool operator< (Word const &other) const
- bool operator== (Word const &other) const

  Equality operator for this and another word.
- bool operator!= (Word const &other) const
- size_t getHash () const

  Get the hash of this word. It does not use words original form or annotation,.
- size_t size () const

  Get the number of attributes of the word.
- unsigned long getAttribute (size_t idx) const

  Get a particular attribute value at positition idx.
- std::string toSeparateLines () const

  Get a representation of the word usable as a GraphViz node label.
- std::vector< Word > getGeneralizations () const

  Get a vector of words, containing all generalizations of this word.
- std::string toString ()

  toString

Static Public Member Functions

- static StringIndex * getStringIndex ()

  Get the pointer to the underlaying string and unique integer management StringIndex.

Private Attributes

- unsigned long attributes [MAX_WORD_ATTRIBUTES]

  Sorted array containing the word attributes.
- size_t numAttributes

  Integer storing the attribute count. This has to be updated explicitly.
- unsigned long original

  Integer representing the original string in the corpus (not the lemma).
- unsigned long lemma
- unsigned long annotation

  Integer representing the annotation of the word.

Static Private Attributes

- static StringIndex index

  StringIndex that takes care string to integer and vice versa management.

Friends

- std::ostream & operator<< (std::ostream &os, Word const &word)
- std::istream & operator>> (std::istream &is, Word &word)

6.36.1 Detailed Description

Word represents attributes of a single word or a pattern.

If the word is in a sentence, it also has information about its original representation form and a possible annotation. Words in patterns do not have any meaningful original representation or annotation.

Note that word attributes are not real strings, but instead unique integers, where each integer represents a single string. The string to int management and vice versa are done by

See Also

StringIndex class. These integers are stored in a sorted array.

6.36.2 Constructor & Destructor Documentation

6.36.2.1 pfe::Word::Word ( )

Default constructor. Creates a default empty word.

6.36.3 Member Function Documentation

6.36.3.1 void pfe::Word::addAttribute (std::string const & *attribute*)

Add an attribute string to the word.

Parameters

attribute,:	String to be added, either lemma or a explicit attribute string.

6.36.3.2 void pfe::Word::addAttribute (unsigned long const *attribute,* bool *toEnd* = `false`)

Add an attribute to the word.

Parameters

attribute,:	unique integer representing a string. No checking is done, if it is a valid integer.
toEnd,:	indicate, if the attribute should be added to the end of word attribute array. Use this only, if you are inserting attributes in sorted order. By default, leave it to false.

6.36.3.3 unsigned long pfe::Word::getAnnotation ( ) const

Get the unique integer that declares the annotation for the word.

6.36.3.4 unsigned long pfe::Word::getAttribute (size_t *idx*) const [inline]

Get a particular attribute value at posititon idx.

6.36.3.5 std::vector< Word > pfe::Word::getGeneralizations ( ) const

Get a vector of words, containing all generalizations of this word.

6.36.3.6 size_t pfe::Word::getHash ( ) const

Get the hash of this word. It does not use words original form or annotation,.

6.36.3.7 unsigned long pfe::Word::getLemma ( ) const

Get the lemma ID.

6.36.3.8 unsigned long pfe::Word::getOriginal ( ) const

Get the unique integer that represents the original string of the word.

6.36.3.9 static StringIndex* pfe::Word::getStringIndex ( ) [inline], [static]

Get the pointer to the underlaying string and unique integer management StringIndex.

6.36.3.10 Word pfe::Word::LCA (Word const & *other*) const

Given this word and another word, create a new word, that is their least common ancestor. This 'least' means that for words w1, w2, LCA(w1, w2) matches both w1 and w2 and there is no other word w3, such that w3 matches w1 and w2, and LCA(w1, w2) matches w3.

6.36.3.11 bool pfe::Word::matches (Word const & *other*) const

Check, if this word is more generic than the other word.

Returns

true: If this word matches the other word, false otherwise.

6.36.3.12 bool pfe::Word::operator!= (Word const & *other*) const

6.36.3.13 bool pfe::Word::operator< (Word const & *other*) const

Comparison operator for the word. The comparison does not have any particular meaning, but it can be used to store words in a ordered container. The comparison does not use words original representation nor the annotation.

6.36.3.14 bool pfe::Word::operator== (Word const & *other*) const

Equality operator for this and another word.

6.36.3.15 void pfe::Word::setAnnotation (std::string const & *annotation*)

Set the string that declares the annotation for the word.

6.36.3.16 void pfe::Word::setLemma (std::string const & *lemma*)

Set the lemma of the word.

6.36.3.17 void pfe::Word::setOriginal (std::string const & *original*)

Set the string that represents the original representation of the word.

6.36.3.18 size_t pfe::Word::size ( ) const [inline]

Get the number of attributes of the word.

6.36.3.19 std::string pfe::Word::toSeparateLines ( ) const

Get a representation of the word usable as a GraphViz node label.

6.36.3.20 std::string pfe::Word::toString ( ) [inline]

toString

6.36.4 Friends And Related Function Documentation

6.36.4.1 std::ostream& operator<< (std::ostream & *os,* Word const & *word*) [friend]

6.36.4.2 std::istream& operator>> (std::istream & *is,* Word & *word*) [friend]

6.36.5 Member Data Documentation

6.36.5.1 unsigned long pfe::Word::annotation [private]

Integer representing the annotation of the word.

6.36.5.2 unsigned long pfe::Word::attributes[MAX_WORD_ATTRIBUTES] [private]

Sorted array containing the word attributes.

6.36.5.3 StringIndex pfe::Word::index [static], [private]

StringIndex that takes care string to integer and vice versa management.

6.36.5.4 unsigned long pfe::Word::lemma [private]

Integer representing the lemma of the string in the corpus. Do not use it for matching tasks, its purpose is only reconstruction of the original coprus and other task that require lemma.

6.36.5.5 size_t pfe::Word::numAttributes [private]

Integer storing the attribute count. This has to be updated explicitly.

6.36.5.6 unsigned long pfe::Word::original [private]

Integer representing the original string in the corpus (not the lemma).

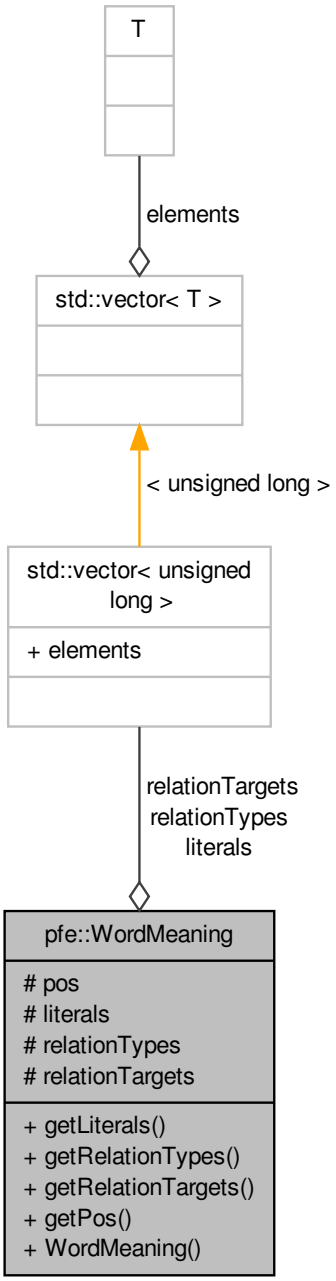
The documentation for this class was generated from the following files:

- include/Corpus.hpp
- src/Corpus.cpp

6.37 pfe::WordMeaning Struct Reference

```
#include <Wordnet.hpp>
```

Collaboration diagram for pfe::WordMeaning:



Public Member Functions

- std::vector< unsigned long > const & getLiterals () const
- std::vector< unsigned long > const & getRelationTypes () const
- std::vector< unsigned long > const & getRelationTargets () const

- WordnetPOS getPos () const
- WordMeaning ()

Protected Attributes

- WordnetPOS pos
- std::vector< unsigned long > literals
- std::vector< unsigned long > relationTypes
- std::vector< unsigned long > relationTargets

Friends

- class Wordnet
- class WordnetParser

6.37.1 Detailed Description

WordMeaning is a concept in Wordnet, that lists all synonymous lemmata and contains relations to other WordMeaning instances.

6.37.2 Constructor & Destructor Documentation

6.37.2.1 pfe::WordMeaning::WordMeaning ( ) [inline]

6.37.3 Member Function Documentation

6.37.3.1 std::vector<unsigned long> const& pfe::WordMeaning::getLiterals ( ) const [inline]

6.37.3.2 WordnetPOS pfe::WordMeaning::getPos ( ) const [inline]

6.37.3.3 std::vector<unsigned long> const& pfe::WordMeaning::getRelationTargets ( ) const [inline]

6.37.3.4 std::vector<unsigned long> const& pfe::WordMeaning::getRelationTypes ( ) const [inline]

6.37.4 Friends And Related Function Documentation

6.37.4.1 friend class Wordnet [friend]

6.37.4.2 friend class WordnetParser [friend]

6.37.5 Member Data Documentation

6.37.5.1 std::vector<unsigned long> pfe::WordMeaning::literals [protected]

6.37.5.2 WordnetPOS pfe::WordMeaning::pos [protected]

6.37.5.3 std::vector<unsigned long> pfe::WordMeaning::relationTargets [protected]

6.37.5.4 std::vector<unsigned long> pfe::WordMeaning::relationTypes [protected]

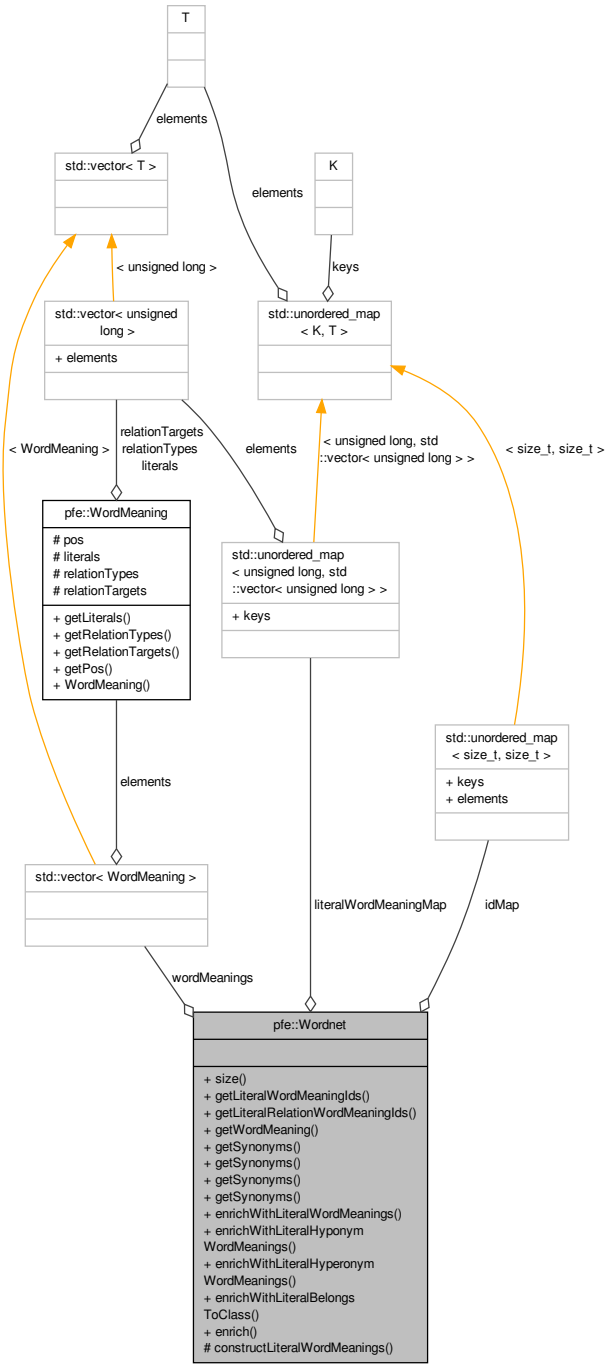
The documentation for this struct was generated from the following file:

- include/Wordnet.hpp

6.38 pfe::Wordnet Class Reference

`#include <Wordnet.hpp>`

Collaboration diagram for pfe::Wordnet:



Public Member Functions

- size_t size () const

- std::vector< unsigned long >
  const & getLiteralWordMeaningIds (unsigned long literalId)

  Retrieve all WordMeaning ids, that contain given literal.
- std::vector< unsigned long > getLiteralRelationWordMeaningIds (unsigned long literalId, WordnetRelations rel)
- WordMeaning const & getWordMeaning (unsigned long id)

  Get the reference to the WordMeaining instance with given id.
- std::vector< unsigned long > getSynonyms (unsigned long literal)

  Get the synonyms for the given literal.
- std::vector< unsigned long > getSynonyms (unsigned long literal, WordnetPOS posTag)
- std::vector< unsigned long > getSynonyms (std::string literal)
- std::vector< unsigned long > getSynonyms (std::string literal, WordnetPOS posTag)
- void enrichWithLiteralWordMeanings (Corpus &corpus)
- void enrichWithLiteralHyponymWordMeanings (Corpus &corpus)
- void enrichWithLiteralHyperonymWordMeanings (Corpus &corpus)
- void enrichWithLiteralBelongsToClass (Corpus &corpus)
- void enrich (Corpus &corpus)

Protected Member Functions

- void constructLiteralWordMeanings ()

Protected Attributes

- std::unordered_map< size_t,
  size_t > idMap
- std::vector< WordMeaning > wordMeanings
- std::unordered_map< unsigned
  long, std::vector< unsigned
  long > > literalWordMeaningMap

Friends

- class WordnetParser

6.38.1 Member Function Documentation

6.38.1.1 void pfe::Wordnet::constructLiteralWordMeanings ( ) [protected]

6.38.1.2 void pfe::Wordnet::enrich (Corpus & *corpus*)

6.38.1.3 void pfe::Wordnet::enrichWithLiteralBelongsToClass (Corpus & *corpus*)

6.38.1.4 void pfe::Wordnet::enrichWithLiteralHyperonymWordMeanings (Corpus & *corpus*)

6.38.1.5 void pfe::Wordnet::enrichWithLiteralHyponymWordMeanings (Corpus & *corpus*)

6.38.1.6 void pfe::Wordnet::enrichWithLiteralWordMeanings (Corpus & *corpus*)

6.38.1.7 std::vector< unsigned long > pfe::Wordnet::getLiteralRelationWordMeaningIds (unsigned long *literalId,* WordnetRelations *rel*)

Return all WordMeaning ids, that are targets for any WordMeaining containing given literal for given relation.

6.38.1.8 std::vector<unsigned long> const& pfe::Wordnet::getLiteralWordMeaningIds (unsigned long *literalId*) [inline]

Retrieve all WordMeaning ids, that contain given literal.

6.38.1.9 std::vector< unsigned long > pfe::Wordnet::getSynonyms (unsigned long *literal*)

Get the synonyms for the given literal.

6.38.1.10 std::vector< unsigned long > pfe::Wordnet::getSynonyms (unsigned long *literal,* WordnetPOS *posTag*)

6.38.1.11 std::vector< unsigned long > pfe::Wordnet::getSynonyms (std::string *literal*)

6.38.1.12 std::vector< unsigned long > pfe::Wordnet::getSynonyms (std::string *literal,* WordnetPOS *posTag*)

6.38.1.13 WordMeaning const& pfe::Wordnet::getWordMeaning (unsigned long *id*) [inline]

Get the reference to the WordMeaining instance with given id.

6.38.1.14 size_t pfe::Wordnet::size ( ) const [inline]

6.38.2 Friends And Related Function Documentation

6.38.2.1 friend class WordnetParser [friend]

6.38.3 Member Data Documentation

6.38.3.1 std::unordered_map<size_t, size_t> pfe::Wordnet::idMap [protected]

6.38.3.2 std::unordered_map<unsigned long, std::vector<unsigned long> > pfe::Wordnet::literalWordMeaningMap [protected]

6.38.3.3 std::vector<WordMeaning> pfe::Wordnet::wordMeanings [protected]

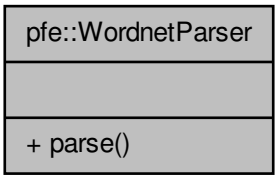
The documentation for this class was generated from the following files:

- include/Wordnet.hpp
- src/Wordnet.cpp

6.39 pfe::WordnetParser Class Reference

```
#include <Wordnet.hpp>
```

Collaboration diagram for pfe::WordnetParser:



Static Public Member Functions

- static Wordnet parse (std::string filename)

6.39.1 Member Function Documentation

6.39.1.1 Wordnet pfe::WordnetParser::parse (std::string *filename*) [static]

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

- include/Wordnet.hpp
- src/Wordnet.cpp

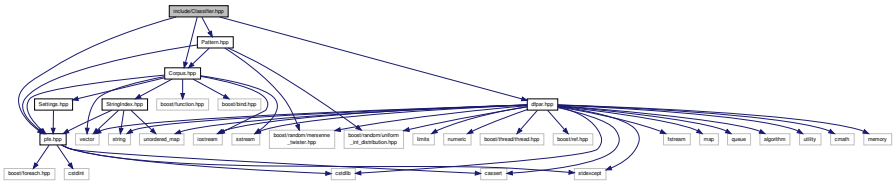
Chapter 7

File Documentation

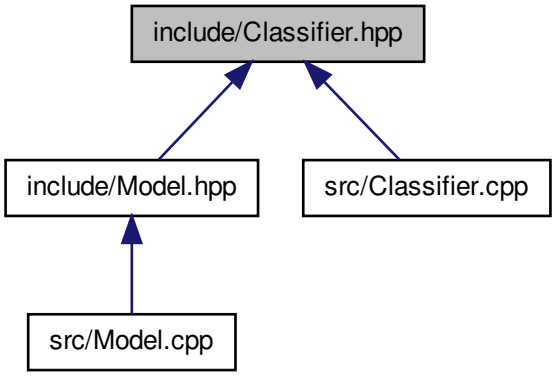
7.1 include/Classifier.hpp File Reference

```
#include <pfe.hpp>
#include <Corpus.hpp>
#include <Pattern.hpp>
#include <dfpar.hpp>
```

Include dependency graph for Classifier.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class pfe::PatternClassifier
- class pfe::RFClassifier

  Random forest classifier.

Namespaces

- namespace pfe

Typedefs

- typedef std::pair
  < CoverElement, double > pfe::CoverElementProb
- typedef std::vector
  < CoverElementProb > pfe::ElementProbVector

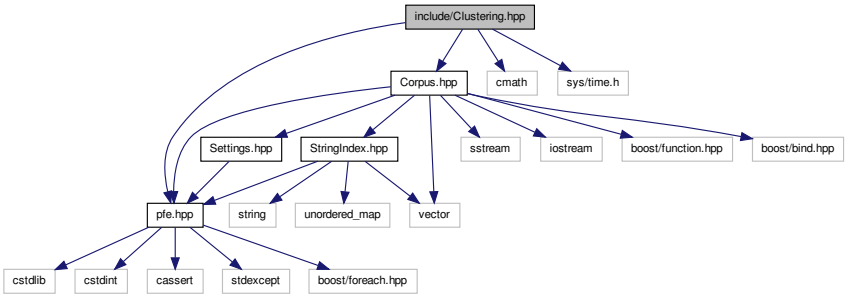
Functions

- Corpus pfe::mapToCorpus (Corpus const &corpus, ElementProbVector const &v, std::string const &annot, double treshold=0.5)
- Cover pfe::filterElements (ElementProbVector const &v, double treshold=0.5)

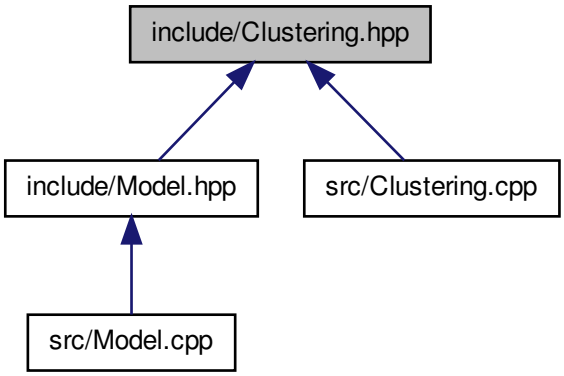
7.2 include/Clustering.hpp File Reference

```
#include <pfe.hpp>
#include <Corpus.hpp>
#include <cmath>
#include <sys/time.h>
```

Include dependency graph for Clustering.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class pfe::VectorSpaceModel
- class pfe::Clusterer

Namespaces

- namespace pfe

Functions

- std::vector< size_t > pfe::kmedoidsIndices (const Corpus &corpus, size_t const K)

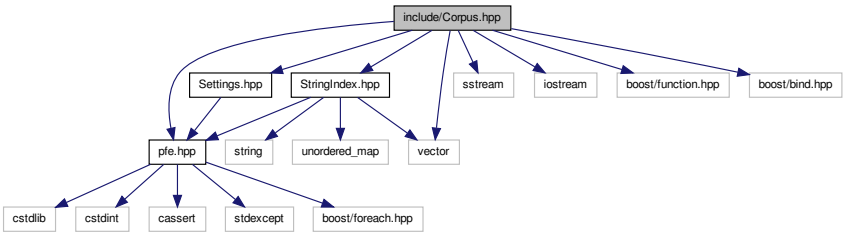
  Simple clustering functions ///.
- std::vector< Corpus > pfe::kmedoids (const Corpus &corpus, size_t const K)

  Kmedoids clustering function that cluster given corpus into K smaller corpora.

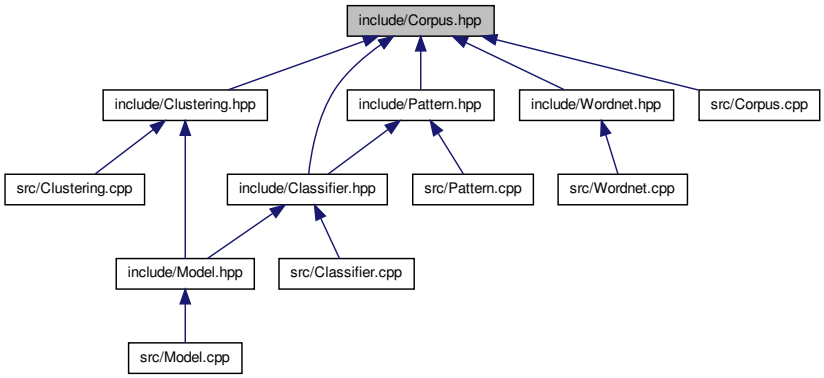
7.3 include/Corpus.hpp File Reference

```
#include <pfe.hpp>
#include <Settings.hpp>
#include <StringIndex.hpp>
#include <vector>
#include <sstream>
#include <iostream>
#include <boost/function.hpp>
#include <boost/bind.hpp>
```

Include dependency graph for Corpus.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class pfe::Word
- class pfe::CorpusParser

  Class for reading corpus from PFE format into memory.
- struct pfe::CoverElement
- struct std::hash< pfe::Word >

  Specialize hash function for word class.
- struct std::equal_to< pfe::Word >

  Specialize equal_to function for word class.
- struct std::hash< pfe::CoverElement >

  Specialize hash function for CoverElement class.
- struct std::equal_to< pfe::CoverElement >

  Specialize equal_to function for CoverElement class.
- class pfe::Cover
- class pfe::CorpusIndex

  CorpusIndex class helps to calculate covers of different words in a corpus efficiently.

Namespaces

- namespace pfe

Typedefs

- typedef std::vector< CoverElement > pfe::CoverElementList
- typedef std::vector< Cover > pfe::CoverVector
- typedef std::vector< Word > pfe::Sentence
- typedef std::vector< Sentence > pfe::Corpus

Functions

- unsigned long pfe::getID (std::string const &attribute)

  Global function to get attribute string from attribute ID.
- unsigned long pfe::getID (const char *const attribute)
- std::string pfe::getString (unsigned long const ID)

  Global function to get string from attribute ID.
- std::unordered_map< unsigned long, size_t > pfe::getAnnotCounts (Corpus const &corpus)

  Get a unordered map of annotation counts from the corpus.
- std::string pfe::toString (Sentence::const_iterator begin, Sentence::const_iterator end)
- std::string pfe::toString (Sentence const &sentence, size_t begin, size_t end)
- std::string pfe::toString (Sentence const &sentence)
- Corpus pfe::randomCorpusSample (Corpus corpus, size_t k)

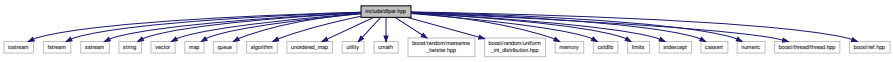
  Make a random sample of given corpus of size k.
- std::ostream & pfe::printCoverElementLemmata (CoverElement const &elem, Corpus const &corpus, std::ostream &os)
- std::ostream & pfe::printCoverElementOriginal (CoverElement const &elem, Corpus const &corpus, std::ostream &os)
- std::ostream & pfe::printCoverLemmata (Cover const &cover, Corpus const &corpus, std::ostream &os)
- std::ostream & pfe::printCoverOriginal (Cover const &cover, Corpus const &corpus, std::ostream &os)
- template<typename T >
  std::vector< T > pfe::set_union (std::vector< T > const &A, std::vector< T > const &B)
- template<typename T >
  std::vector< T > pfe::set_intersection (std::vector< T > const &A, std::vector< T > const &B)
- template<typename T >
  std::vector< T > pfe::set_difference (std::vector< T > const &A, std::vector< T > const &B)

7.4 include/dfpar.hpp File Reference

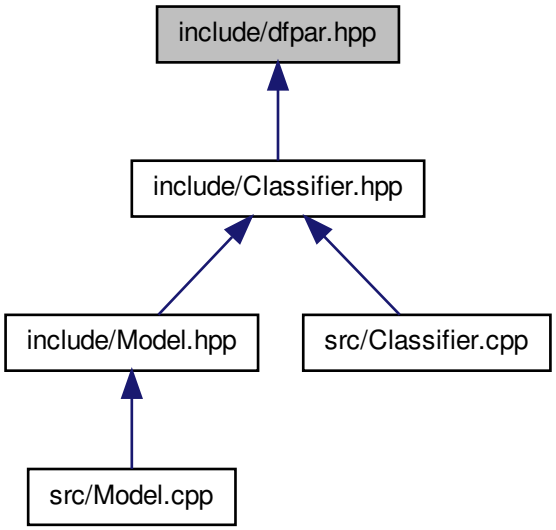
```
#include <iostream>
```

```
#include <fstream>
#include <sstream>
#include <string>
#include <vector>
#include <map>
#include <queue>
#include <algorithm>
#include <unordered_map>
#include <utility>
#include <cmath>
#include <boost/random/mersenne_twister.hpp>
#include <boost/random/uniform_int_distribution.hpp>
#include <memory>
#include <cstdlib>
#include <limits>
#include <stdexcept>
#include <cassert>
#include <numeric>
#include <boost/thread/thread.hpp>
#include <boost/ref.hpp>
```

Include dependency graph for dfpar.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class DForest< T, L >

- class DTree< T, L >
- class DNode< T, L >
- class DataSet< T, L >
- struct ResultRow< L >
- class DTreeBuilder< T, L >
- class DForestBuilder< T, L >

Macros

- #define DFPAR_USE_BOOST_THREADS
- #define DFPAR_USE_BOOST_RANDOM
- #define dfparref boost::ref
- #define DFPAR_VERSION "1.0"
- #define DFPAR_BEGIN_NAMESPACE namespace dfpar {
- #define DFPAR_END_NAMESPACE }

Typedefs

- typedef boost::thread dfparthread

Functions

- template<typename T >
  std::vector< T > sample (std::vector< T >, size_t const k)
- template<typename T >
  std::vector< T > createRange (T begin, T end)
- double entropy (std::vector< size_t > const &, size_t total=0)
- double gain (std::vector< size_t > const &, std::vector< size_t > const &)
- template<typename T >
  bool isDiscrete ()
- template<>
  bool isDiscrete< float > ()
- template<>
  bool isDiscrete< double > ()
- template<>
  bool isDiscrete< long double > ()
- template<typename T >
  T defaultValue ()
- template<>
  int defaultValue< int > ()
- template<>
  unsigned int defaultValue< unsigned int > ()
- template<>
  float defaultValue< float > ()
- template<>
  double defaultValue< double > ()
- template<>
  long double defaultValue< long double > ()
- template<>
  std::string defaultValue< std::string > ()
- template<typename T >
  bool discreteEqual (T const &x1, T const &x2)
- template<typename T >
  bool continuousEqual (T const &x1, T const &x2)

- template<typename T >
  bool equal (T const &x1, T const &x2)
- template<>
  bool equal< float > (float const &x1, float const &x2)
- template<>
  bool equal< double > (double const &x1, double const &x2)
- template<>
  bool equal< long double > (long double const &x1, long double const &x2)
- size_t gainHelper (size_t count, size_t total)

7.4.1 Macro Definition Documentation

7.4.1.1 #define DFPAR_BEGIN_NAMESPACE namespace dfpar {

7.4.1.2 #define DFPAR_END_NAMESPACE }

7.4.1.3 #define DFPAR_USE_BOOST_RANDOM

7.4.1.4 #define DFPAR_USE_BOOST_THREADS

7.4.1.5 #define DFPAR_VERSION "1.0"

7.4.1.6 #define dfparref boost::ref

7.4.2 Typedef Documentation

7.4.2.1 typedef boost::thread dfparthread

7.4.3 Function Documentation

7.4.3.1 template<typename T > bool continuousEqual (T const & *x1*, T const & *x2*) [inline]

7.4.3.2 template<typename T > std::vector< T > createRange (T *begin*, T *end*)

Create a range [begin, end) of type T. the operator++ is used to increment from the begin value greater or equal to end is reached.

Parameters

begin	The first value of the range.
end	The first value excluded from the range.

Returns

The vector<T> instance containing the range.

7.4.3.3 template<typename T > T defaultValue ( ) [inline]

7.4.3.4 template<> double defaultValue< double > ( ) [inline]

7.4.3.5 template<> float defaultValue< float > ( ) [inline]

7.4.3.6 template<> int defaultValue< int > ( ) [inline]

7.4.3.7 template<> long double defaultValue< long double > ( ) [inline]

7.4.3.8 template<> std::string defaultValue< std::string > ( ) [inline]

7.4.3.9 template<> unsigned int defaultValue< unsigned int > ( ) [inline]

7.4.3.10 template<typename T > bool discreteEqual (T const & *x1*, T const & *x2*) [inline]

7.4.3.11 double entropy (std::vector< size_t > const & *counts*, size_t *total*) [inline]

Calculate the entropy.

Parameters

counts	The vector of class counts.
total	Optional sum of the counts, if not given, the function will calculate it itself.

Returns

Entropy.

7.4.3.12 template<typename T > bool equal (T const & *x1*, T const & *x2*) [inline]

7.4.3.13 template<> bool equal< double > (double const & *x1*, double const & *x2*) [inline]

7.4.3.14 template<> bool equal< float > (float const & *x1*, float const & *x2*) [inline]

Function for comparing if values x1 and x2 of type T are (almost) equal.

Template Parameters

T	Type of the values to compare.

Parameters

x1	The first value.
x2	The second value.

Returns

true if values are (almost) equal and false otherwise.

7.4.3.15 template<> bool equal< long double > (long double const & *x1*, long double const & *x2*) [inline]

7.4.3.16 double gain (std::vector< size_t > const & *counts*, std::vector< size_t > const & *totals*) [inline]

Calculate information gain of splitting the the classes described by totals into two, with given respective counts of the first split.

Parameters

counts	The counts of classes in the first split.
totals	The total number of counts of classes (in the same order) in both splits.

Returns

The information gain by splitting the samples into two.

7.4.3.17 size_t gainHelper (size_t *count*, size_t *total*) [inline]

7.4.3.18 template<typename T > bool isDiscrete ( ) [inline]

Function for determining, if a type is discrete. By default, each type is reported as discrete, except float, double and long double.

7.4.3.19 template<> bool isDiscrete< double > ( ) [inline]

7.4.3.20 template<> bool isDiscrete< float > ( ) [inline]

7.4.3.21 template<> bool isDiscrete< long double > ( ) [inline]

7.4.3.22 template<typename T > std::vector< T > sample (std::vector< T > *values*, size_t const *k*)

Function for taking a random sample without replacement.

Parameters

values	The vector of values describing the population.
k	The sample size.

Template Parameters

T	the type of the vector elements.

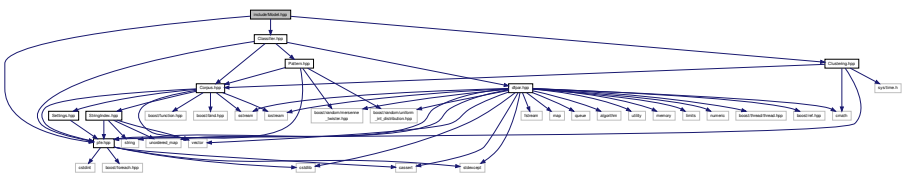
Returns

Vector containing a sample from the population.

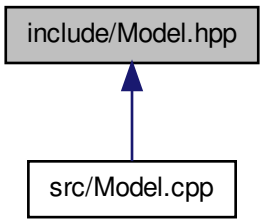
7.5 include/Model.hpp File Reference

```
#include <pfe.hpp>
#include <Classifier.hpp>
#include <Clustering.hpp>
```

Include dependency graph for Model.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- struct pfe::ModelBuildSettings

  Class representing possible settings for building a model.
- class pfe::Model
- class pfe::RFModel
- class pfe::RecPrecModel
- class pfe::ClusterModel
- class pfe::ClusterFeatureModel

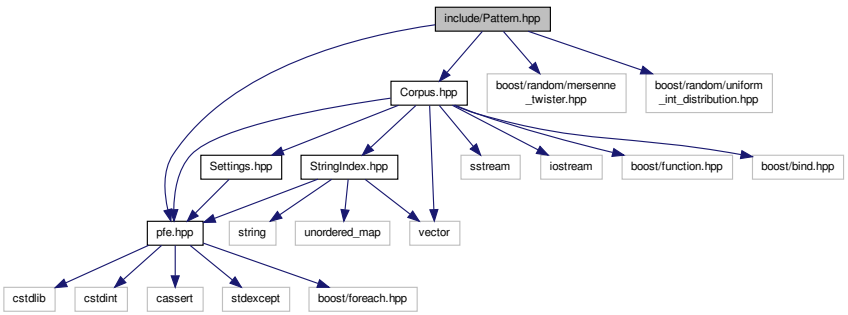
Namespaces

- namespace pfe

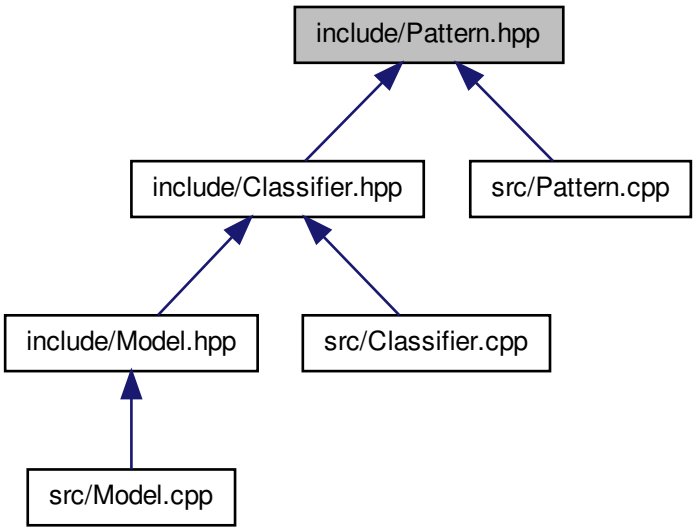
7.6 include/Pattern.hpp File Reference

```
#include <pfe.hpp>
#include <Corpus.hpp>
#include <boost/random/mersenne_twister.hpp>
#include <boost/random/uniform_int_distribution.hpp>
```

Include dependency graph for Pattern.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class pfe::Automaton
- class pfe::Pattern

  Pattern consist of left context, entity area and right context, which are all automatons.
- class pfe::Criteria

  Base class for pattern mining criteria.
- class pfe::GoodPrecisionCriteria

  Criteria, that maximizes precision.
- class pfe::GoodRecallCriteria

  Criteria, that maximizer recall.
- class pfe::MixedCriteria

  Criteria, that tries sums the precision and recall as weight.
- class pfe::UniquenessCriteria

  Criteria, that tries to maximize uniqueness.
- class pfe::RandomGeneralizer

  Base class for random generalizers for optimisation tasks.
- class pfe::RandomAttributeGeneralizer

Namespaces

- namespace pfe

Typedefs

- typedef std::vector< double > pfe::DoubleVector

- typedef std::vector< Automaton > pfe::AutomatonVector
- typedef std::vector< Word > pfe::WordVector
- typedef std::vector< unsigned > pfe::WordIndexVector
- typedef std::vector< Pattern > pfe::PatternVector

Enumerations

- enum pfe::LCATrim { pfe::LCATrimNone, pfe::LCATrimLeft, pfe::LCATrimRight }

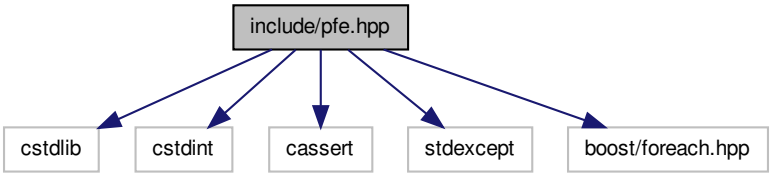
Functions

- void pfe::renderPatterns (PatternVector const &patterns, std::string path, std::string title)
- void pfe::randomSearch (PatternVector const &patterns, CorpusIndex &index, Cover const &trueCover, size_t N, Criteria *criteria, RandomGeneralizer *generalizer, PatternVector &resPatterns, CoverVector &resCovers)

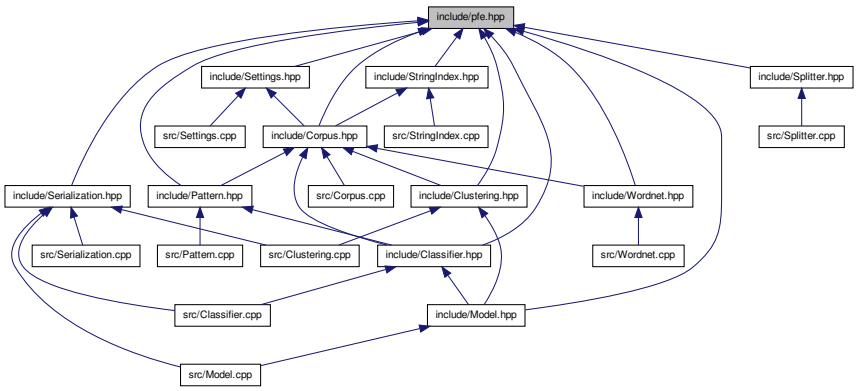
7.7 include/pfe.hpp File Reference

```
#include <cstdlib>
#include <cstdint>
#include <cassert>
#include <stdexcept>
#include <boost/foreach.hpp>
```

Include dependency graph for pfe.hpp:



This graph shows which files directly or indirectly include this file:



Macros

- #define PFE_AUTHORS "Timo Petmanson\nFanny-Dhelia Pajuste\nSven Laur\0"

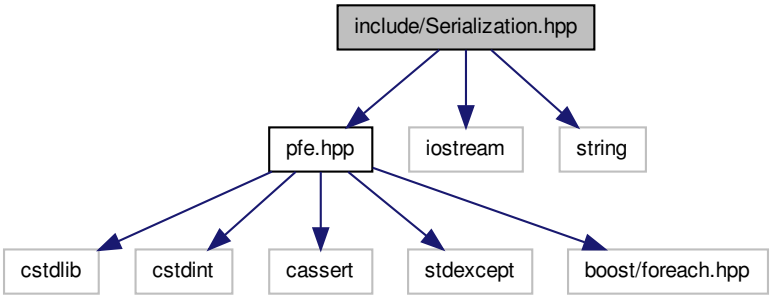
7.7.1 Macro Definition Documentation

7.7.1.1 #define PFE_AUTHORS "Timo Petmanson\nFanny-Dhelia Pajuste\nSven Laur\0"

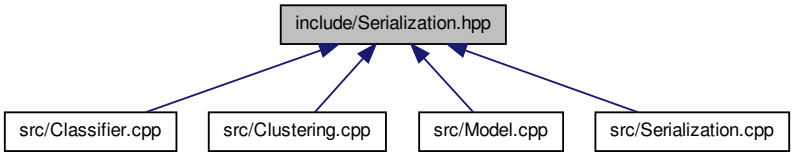
7.8 include/Serialization.hpp File Reference

```
#include <pfe.hpp>
#include <iostream>
#include <string>
```

Include dependency graph for Serialization.hpp:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace pfe

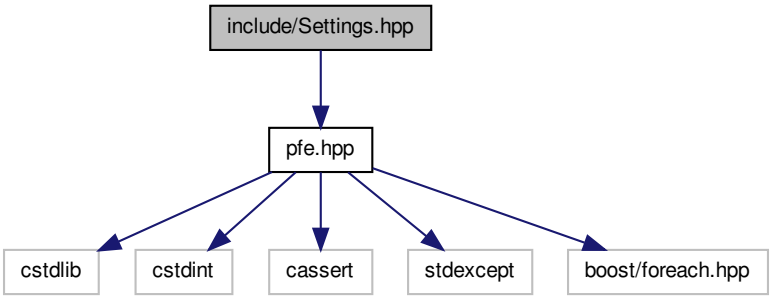
Functions

- void pfe::writeString (std::string const &s, std::ostream &os)
- std::string pfe::readString (std::istream &is)
- void pfe::writeDouble (double value, std::ostream &os)
- double pfe::readDouble (std::istream &is)

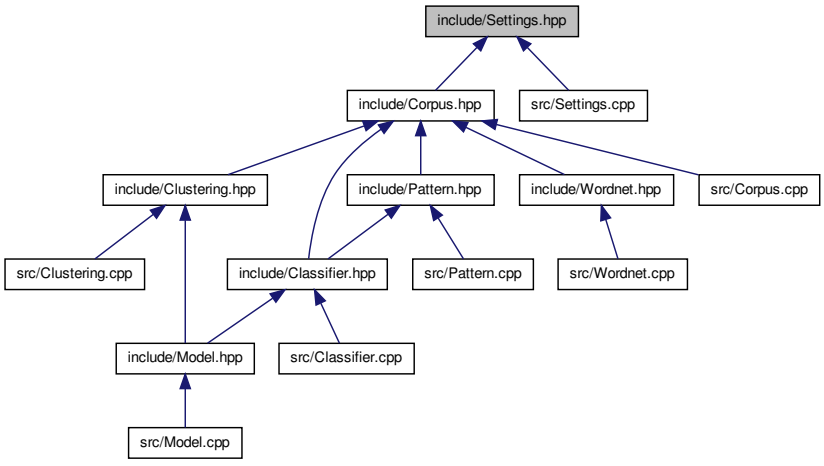
- void pfe::writeUl (unsigned long value, std::ostream &os)
- unsigned long pfe::readUl (std::istream &is)
- void pfe::writeLong (long value, std::ostream &os)
- long pfe::readLong (std::istream &is)

7.9 include/Settings.hpp File Reference

`#include <pfe.hpp>`
Include dependency graph for Settings.hpp:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace pfe

Functions

- int pfe::getNumAvailableCores ()

- std::size_t pfe::getNumThreads ()
- void pfe::setNumThreads (size_t n)
- void pfe::intFunction (int value)
- bool pfe::boolReturnFunction ()

Variables

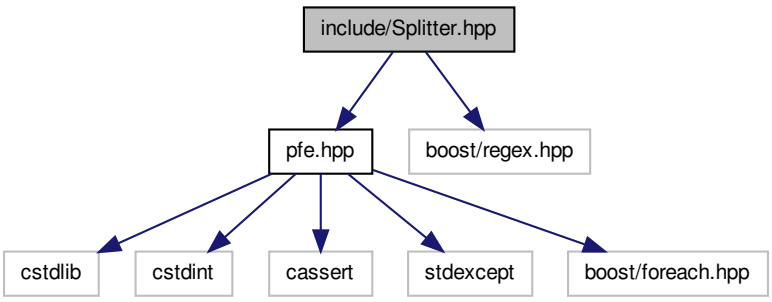
- const size_t pfe::MAX_WORD_ATTRIBUTES = 32
- const size_t pfe::PFE_MAJOR_VERSION = 1

  Library version.
- const size_t pfe::PFE_MINOR_VERSION = 0

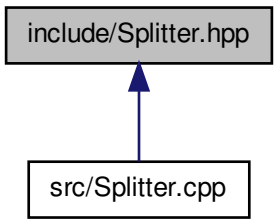
7.10 include/Splitter.hpp File Reference

```
#include <pfe.hpp>
#include <boost/regex.hpp>
```

Include dependency graph for Splitter.hpp:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace pfe

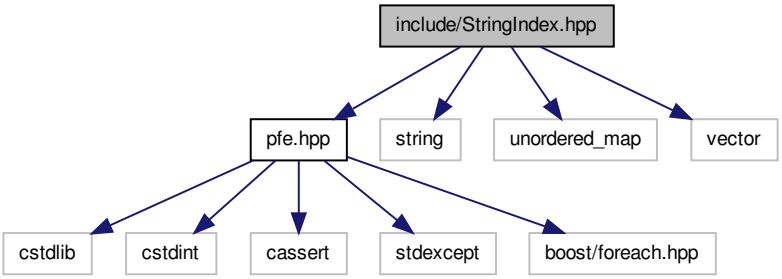
Functions

- void pfe::split_sentences (std::string textf, std::string splitterf, bool signs=false)
- void pfe::split_sentences (std::istream &, std::ostream &, bool signs=false)

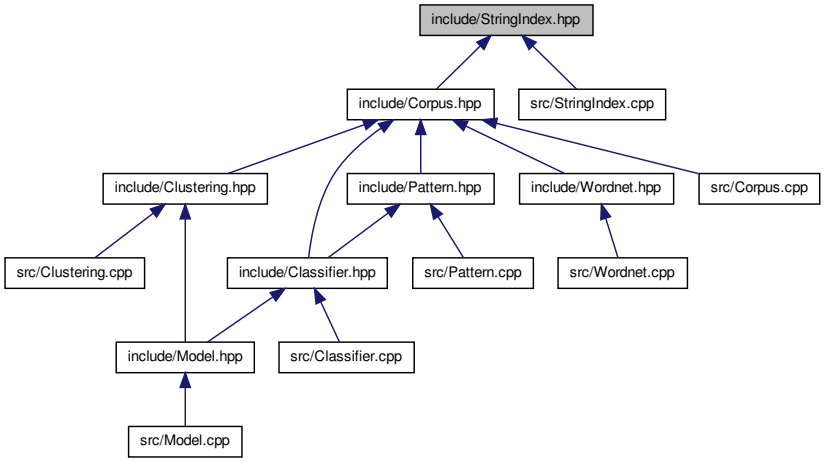
7.11 include/StringIndex.hpp File Reference

```
#include <pfe.hpp>
#include <string>
#include <unordered_map>
#include <vector>
```

Include dependency graph for StringIndex.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class pfe::StringIndex

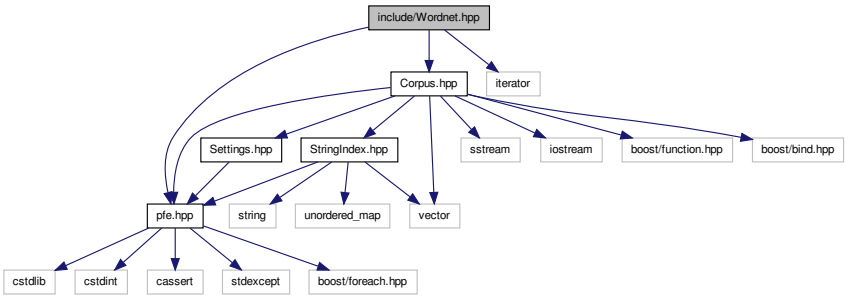
Namespaces

- namespace pfe

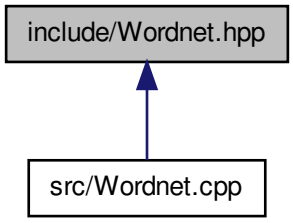
7.12 include/Wordnet.hpp File Reference

```
#include <pfe.hpp>
#include <Corpus.hpp>
#include <iterator>
```

Include dependency graph for Wordnet.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- struct pfe::WordMeaning
- class pfe::Wordnet
- class pfe::WordnetParser

Namespaces

- namespace pfe

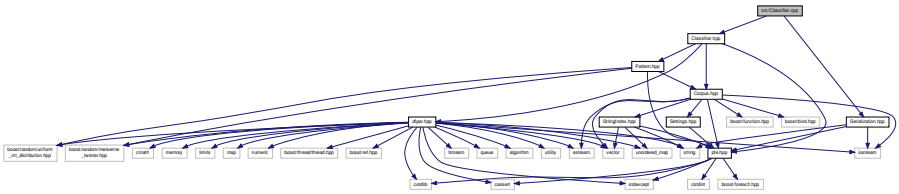
Enumerations

- enum pfe::WordnetRelations { pfe::WORDNET_HAS_HYPERONYM, pfe::WORDNET_HAS_HYPONYM, pfe::WORDNET_BELONGS_TO_CLASS }
- enum pfe::WordnetPOS { pfe::WORDNET_NONE, pfe::WORDNET_VERB, pfe::WORDNET_NOUN, pfe::WORDNET_ADJECTIVE, pfe::WORDNET_ADVERB, pfe::WORDNET_PROPER_NOUN }

7.13 src/Classifier.cpp File Reference

```
#include <Classifier.hpp>
#include <Serialization.hpp>
```

Include dependency graph for Classifier.cpp:



Namespaces

- namespace pfe

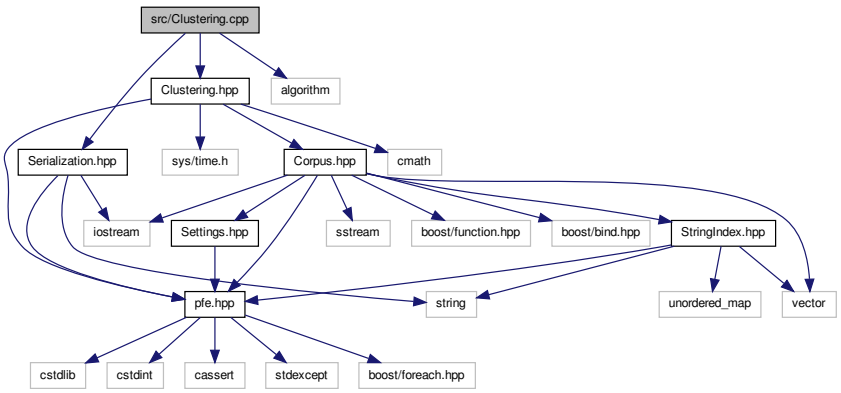
Functions

- std::ostream & pfe::operator<< (std::ostream &os, RFClassifier &classifier)
- std::istream & pfe::operator>> (std::istream &is, RFClassifier &classifier)
- Corpus pfe::mapToCorpus (Corpus const &corpus, ElementProbVector const &v, std::string const &annot, double treshold=0.5)
- Cover pfe::filterElements (ElementProbVector const &v, double treshold=0.5)

7.14 src/Clustering.cpp File Reference

```
#include <Clustering.hpp>
#include <Serialization.hpp>
#include <algorithm>
```

Include dependency graph for Clustering.cpp:



Namespaces

- namespace pfe

Functions

- std::unordered_map< unsigned long, double > pfe::getTfWeights (Sentence const &sentence)
- double pfe::getDistance (std::unordered_map< unsigned long, double > &idf1, std::unordered_map< unsigned long, double > &idf2)
- std::vector< size_t > pfe::initRandomMedoids (size_t K, size_t const N)

  Helper method for clusterer that creates random initial medoids.
- std::vector< std::vector < size_t > > pfe::divideToClusters (VectorSpaceModel &model, Corpus const &corpus, std::vector< size_t > &medoids)

  Helper method for clusterer for dividing sentences to clusters.
- size_t pfe::findBestMedoid (std::vector< size_t > indices, VectorSpaceModel &model)
- std::ostream & pfe::operator<< (std::ostream &os, Clusterer &clusterer)
- std::istream & pfe::operator>> (std::istream &is, Clusterer &clusterer)
- std::vector< size_t > pfe::kmedoidsIndices (const Corpus &corpus, size_t const K)

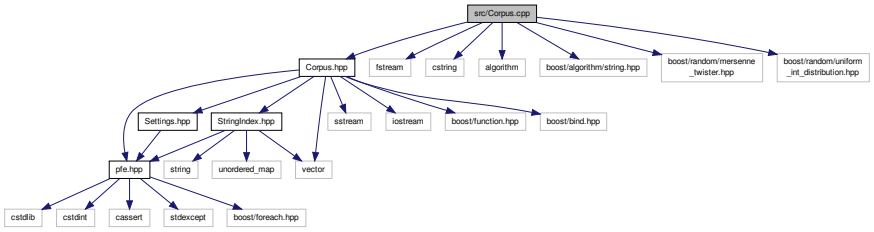
  Simple clustering functions ///.
- std::vector< Corpus > pfe::kmedoids (const Corpus &corpus, size_t const K)

  Kmedoids clustering function that cluster given corpus into K smaller corpora.

7.15 src/Corpus.cpp File Reference

```
#include <Corpus.hpp>
#include <fstream>
#include <cstring>
#include <algorithm>
#include <boost/algorithm/string.hpp>
#include <boost/random/mersenne_twister.hpp>
#include <boost/random/uniform_int_distribution.hpp>
```

Include dependency graph for Corpus.cpp:



Namespaces

- namespace pfe

Typedefs

- typedef std::vector< std::string > pfe::StringVector

Functions

- bool pfe::getBit (uintmax_t value, size_t pos)
- std::ostream & pfe::operator<< (std::ostream &os, Word const &word)
- std::istream & pfe::operator>> (std::istream &is, Word &word)
- unsigned long pfe::getID (std::string const &attribute)

  Global function to get attribute string from attribute ID.
- unsigned long pfe::getID (const char *const attribute)
- std::string pfe::getString (unsigned long const ID)

  Global function to get string from attribute ID.
- std::string pfe::toString (Sentence::const_iterator begin, Sentence::const_iterator end)
- std::string pfe::toString (Sentence const &sentence)
- std::string pfe::toString (Sentence const &sentence, size_t begin, size_t end)
- std::unordered_map< unsigned long, size_t > pfe::getAnnotCounts (Corpus const &corpus)

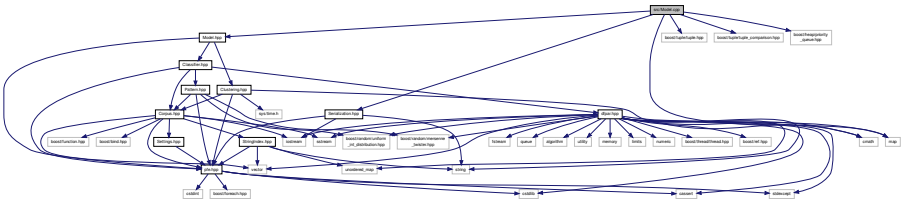
  Get a unordered map of annotation counts from the corpus.
- Corpus pfe::randomCorpusSample (Corpus corpus, size_t k)

  Make a random sample of given corpus of size k.

7.16 src/Model.cpp File Reference

```
#include <Model.hpp>
#include <Serialization.hpp>
#include <boost/tuple/tuple.hpp>
#include <boost/tuple/tuple_comparison.hpp>
#include <boost/heap/priority_queue.hpp>
#include <map>
```

Include dependency graph for Model.cpp:



Namespaces

- namespace pfe

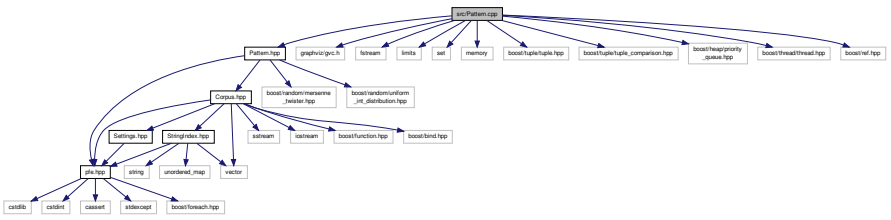
Functions

- std::ostream & pfe::operator<< (std::ostream &os, ModelBuildSettings &settings)
- std::ostream & pfe::operator<< (std::ostream &os, RFModel &model)
- std::istream & pfe::operator>> (std::istream &is, RFModel &model)
- PatternVector pfe::buildPart (Corpus &trainCorpus, Cover &trainCover, Corpus &testCorpus, Cover &testCover, ModelBuildSettings &settings)
- std::ostream & pfe::operator<< (std::ostream &os, RecPrecModel &model)
- std::istream & pfe::operator>> (std::istream &is, RecPrecModel &model)
- std::ostream & pfe::operator<< (std::ostream &os, ClusterModel &model)
- std::istream & pfe::operator>> (std::istream &is, ClusterModel &model)
- std::ostream & pfe::operator<< (std::ostream &os, ClusterFeatureModel &model)
- std::istream & pfe::operator>> (std::istream &is, ClusterFeatureModel &model)

7.17 src/Pattern.cpp File Reference

```
#include <Pattern.hpp>
#include <graphviz/gvc.h>
#include <fstream>
#include <limits>
#include <set>
#include <memory>
#include <boost/tuple/tuple.hpp>
#include <boost/tuple/tuple_comparison.hpp>
#include <boost/heap/priority_queue.hpp>
#include <boost/thread/thread.hpp>
#include <boost/ref.hpp>
```

Include dependency graph for Pattern.cpp:



Namespaces

- namespace pfe

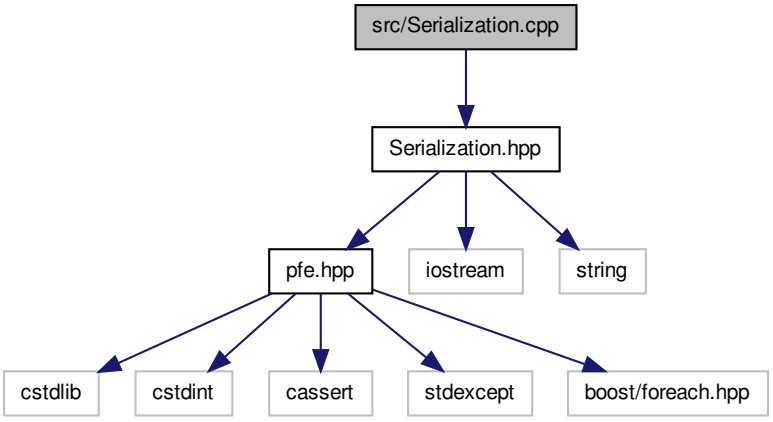
Functions

- std::ostream & pfe::operator<< (std::ostream &os, Automaton const &automaton)
- std::istream & pfe::operator>> (std::istream &is, Automaton &automaton)
- void pfe::patternThreadMatcher (Pattern const &pattern, CorpusIndex &index, Cover &cover, int &completed)
- std::ostream & pfe::operator<< (std::ostream &os, Pattern const &pattern)
- std::istream & pfe::operator>> (std::istream &is, Pattern &pattern)
- void pfe::renderPatterns (PatternVector const &patterns, std::string path, std::string title)
- PatternVector pfe::makeRandomCandidates (PatternVector const &patterns, RandomGeneralizer *generalizer, size_t const N)
- void pfe::randomSearch (PatternVector const &patterns, CorpusIndex &index, Cover const &trueCover, size_t N, Criteria *criteria, RandomGeneralizer *generalizer, PatternVector &resPatterns, CoverVector &resCovers)

7.18 src/Serialization.cpp File Reference

```
#include <Serialization.hpp>
```

Include dependency graph for Serialization.cpp:



Namespaces

- namespace pfe

Functions

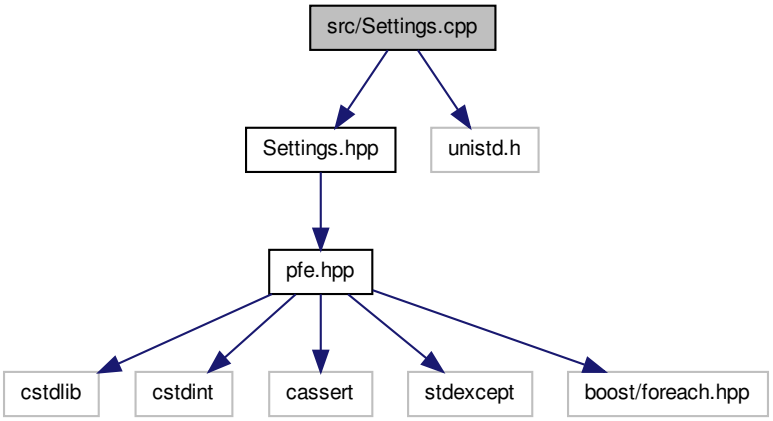
- void pfe::writeString (std::string const &s, std::ostream &os)
- std::string pfe::readString (std::istream &is)
- void pfe::writeDouble (double value, std::ostream &os)
- double pfe::readDouble (std::istream &is)
- void pfe::writeUI (unsigned long value, std::ostream &os)

- unsigned long pfe::readUl (std::istream &is)
- void pfe::writeLong (long value, std::ostream &os)
- long pfe::readLong (std::istream &is)

7.19 src/Settings.cpp File Reference

```
#include <Settings.hpp>
#include <unistd.h>
```

Include dependency graph for Settings.cpp:



Namespaces

- namespace pfe

Functions

- int pfe::getNumAvailableCores ()
- std::size_t pfe::getNumThreads ()
- void pfe::setNumThreads (size_t n)
- void pfe::intFunction (int value)
- bool pfe::boolReturnFunction ()

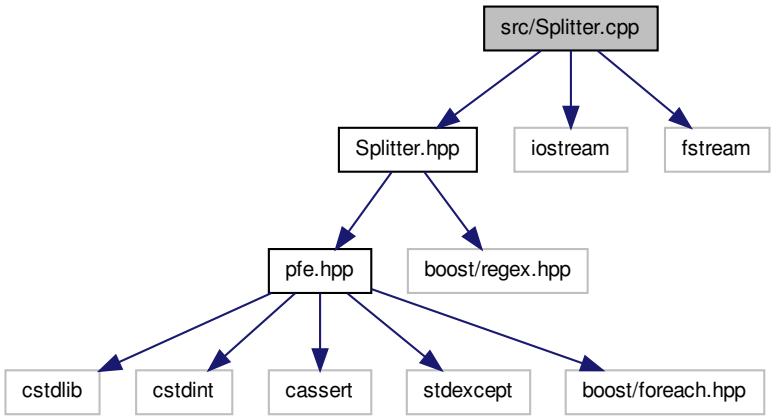
Variables

- size_t pfe::numThreads = 1

7.20 src/Splitter.cpp File Reference

```
#include <Splitter.hpp>
#include <iostream>
#include <fstream>
```

Include dependency graph for Splitter.cpp:



Namespaces

- namespace pfe

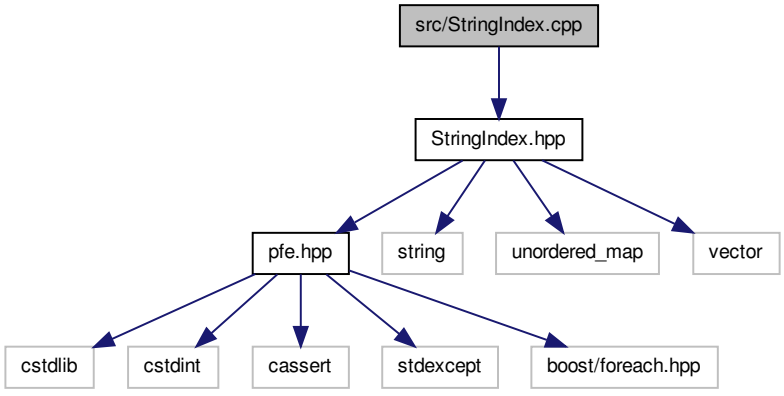
Functions

- void pfe::split_sentences (std::string textf, std::string splitterf, bool signs=false)
- void pfe::split_sentences (std::istream &, std::ostream &, bool signs=false)

7.21 src/StringIndex.cpp File Reference

```
#include <StringIndex.hpp>
```

Include dependency graph for StringIndex.cpp:



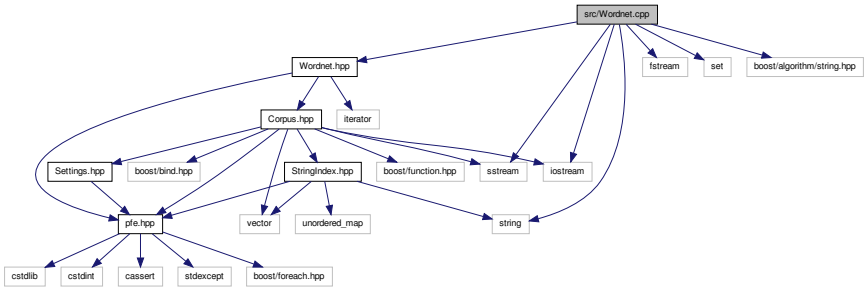
Namespaces

- namespace pfe

7.22 src/Wordnet.cpp File Reference

```
#include <Wordnet.hpp>
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <set>
#include <boost/algorithm/string.hpp>
```

Include dependency graph for Wordnet.cpp:



Namespaces

- namespace pfe