FLAGR

Generated by Doxygen 1.9.5

Chapter 1

(Py)FLAGR

- Fuse, Learn, AGgregate, Rerank

The fusion of multiple ranked lists of elements into a single aggregate list is a well-studied research field with numerous applications in Bioinformatics, recommendation systems, collaborative filtering, election systems and metasearch engines.

FLAGR is a high performance, modular, open source C++ library for rank aggregation problems. It implements baseline and recent state-of-the-art aggregation algorithms that accept ranked preference lists and generate a single consensus list of elements. A portion of these methods apply exploratory analysis techniques and belong to the broad family of unsupervised learning techniques.

PyFLAGR is a Python library built on top of FLAGR library core. It can be easily installed with pip (see below) and used in standard Python programs and Jupyter notebooks. Representative code examples can be found on this notebook.

The current FLAGR version is 1.0.8.

Both libraries are licensed under the Apache License, version 2.

The library is fully documented at https://flagr.site/

1.1 Compiling from Sources

FLAGR can be easily compiled from its C++ sources by using the provided build scripts. The scripts require a working GCC compiler to be installed into the machine that performs the compilation.

There are two build scripts, one for Linux and one for Windows. Specifically:

- In Linux: type `make` in the terminal to build the binaries from the C++ sources. The FLAGR executable file is automatically created into the `bin/Release/` directory of the package. In addition, a shared `.so` library will be created into `bin/` and `pyflagr/pyflagr/`.
- In Windows: type `makefile.bat` in Windows CLI or Windows Powershell. The batch file will build the binaries from the C++ sources and generate `FLAGR.exe` into the `bin/Release/` directory. Moreover, a Dynamic Link `.dll` Library will be created into `bin/` and `pyflagr/pyflagr/`.

1.2 Installing PyFLAGR

PyFLAGR can be installed directly by using `pip`:

```
pip install pyflagr
```

Alternatively, PyFLAGR can be installed from the sources by navigating to the directory where `setup.py` resides:

```
pip install .
```

Chapter 2

Hierarchical 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

Class Documentation

5.1 Aggregator Class Reference

`#include <Aggregator.h>`

Public Member Functions

- Aggregator ()
  Default Constructor.
- ~Aggregator ()
  Destructor.
- class InputList * create_list (char *, double)
  Create a new input list for an aggregator.
- class Voter ** aggregate (char *, class InputParams *)
  Apply the rank aggregation method and construct the final output list.
- void init_weights ()
  Set the initial voter weights.
- void destroy_output_list ()
  Destroy the output list.
- void display ()
- uint16_t get_num_lists ()
  Accessors.
- rank_t get_num_items ()
- class InputList * get_input_list (uint32_t)
- class MergedList * get_output_list ()

5.1.1 Detailed Description

Definition at line 4 of file Aggregator.h.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 Aggregator()

```
Aggregator::Aggregator ( )
```

Default Constructor.

Definition at line 4 of file Aggregator.cpp.

5.1.2.2 ~Aggregator()

```
Aggregator::~Aggregator ( )
```

Destructor.

Definition at line 11 of file Aggregator.cpp.

5.1.3 Member Function Documentation

5.1.3.1 aggregate()

```
class Voter ** Aggregator::aggregate (
            char * topic,
            class InputParams * params )
```

Apply the rank aggregation method and construct the final output list.

Apply the aggregation method of the argument 10X. CombSUM [1]

11X. CombMNZ [1]

1. Condorcet Winners Method
2. Copeland Winners Method
3. The outranking approach of [2]
4. Kemeny Optimal Aggregation (Brute Force Method)
5. The Robust Rank Aggregation algorithm of [7]

5XXX. The DIBRA method of [5]

1. The preference relations method of [3]
2. The weighted agglomerative algorithm of [4]

80X. The Markov Chains methods of [6]

1. The first custom (user-defined) method
2. The second custom (user-defined) method

Definition at line 72 of file Aggregator.cpp.

5.1.3.2 create_list()

```
class InputList * Aggregator::create_list (
            char * v,
            double w )
```

Create a new input list for an aggregator.

Definition at line 28 of file Aggregator.cpp.

5.1.3.3 destroy_output_list()

```
void Aggregator::destroy_output_list ( )
```

Destroy the output list.

Definition at line 64 of file Aggregator.cpp.

5.1.3.4 display()

```
void Aggregator::display ( )
```

Definition at line 159 of file Aggregator.cpp.

5.1.3.5 get_input_list()

```
class InputList * Aggregator::get_input_list (
            uint32_t i )  [inline]
```

Definition at line 176 of file Aggregator.cpp.

5.1.3.6 get_num_items()

```
rank_t Aggregator::get_num_items ( )  [inline]
```

Definition at line 175 of file Aggregator.cpp.

5.1.3.7 get_num_lists()

```
uint16_t Aggregator::get_num_lists ( )  [inline]
```

Accessors.

Definition at line 174 of file Aggregator.cpp.

5.1.3.8 get_output_list()

```
class MergedList * Aggregator::get_output_list ( )  [inline]
```

Definition at line 177 of file Aggregator.cpp.

5.1.3.9 init_weights()

```
void Aggregator::init_weights ( )
```

Set the initial voter weights.

Definition at line 57 of file Aggregator.cpp.

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

- FLAGR/src/Aggregator.h
- FLAGR/src/Aggregator.cpp

5.2 Evaluator Class Reference

```
#include <Evaluator.h>
```

Public Member Functions

- Evaluator ()
  Default Constructor.
- ~Evaluator ()
  Destructor.
- void insert_relev (char *, uint32_t)
  Insert a relevance judgement into the relevs lexicon.
- void clear ()
- void evaluate (rank_t, char *, class MergedList *, FILE *)
  Evaluate a MergedList.
- double evaluate_input (class InputList *)
  Evaluate an InputList.
- void display_relevs ()
- uint32_t get_num_rel ()
  Accessors.
- uint32_t get_true_positives ()
- double get_precision (uint32_t)
- double get_recall (uint32_t)
- double get_F1 (uint32_t)
- double get_dcg (uint32_t)
- double get_ndcg (uint32_t)
- double get_average_precision ()
- double get_average_recall ()
- double get_average_dcg ()
- double get_average_ndcg ()

5.2.1 Detailed Description

Definition at line 4 of file Evaluator.h.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 Evaluator()

```
Evaluator::Evaluator ( )
```

Default Constructor.

Definition at line 4 of file Evaluator.cpp.

5.2.2.2 ~Evaluator()

```
Evaluator::~Evaluator ( )
```

Destructor.

Definition at line 17 of file Evaluator.cpp.

5.2.3 Member Function Documentation

5.2.3.1 clear()

```
void Evaluator::clear ( )
```

Definition at line 32 of file Evaluator.cpp.

5.2.3.2 display_relevs()

```
void Evaluator::display_relevs ( )
```

Definition at line 333 of file Evaluator.cpp.

5.2.3.3 evaluate()

```
void Evaluator::evaluate (
            rank_t ev_pts,
            char * qry,
            class MergedList * lst,
            FILE * eval_file )
```

Evaluate a MergedList.

Create the ideal ranking for the calculation of nDCG

Precision@k

Recall@k

DCG@k

IdealDCG temp: needs to be sorted

Sort ideal_dcg in decreasing relevance order and obtain IdealDCG@k

Compute nDCG@k = DCG@k / IdealDCG@k

Compute Average Precision, Recall, DCG, and nDCG

Write to file

CSV data

Definition at line 55 of file Evaluator.cpp.

5.2.3.4 evaluate_input()

```
double Evaluator::evaluate_input (
            class InputList * lst )
```

Evaluate an InputList.

Create the ideal ranking for the calculation of nDCG

Precision@k

Recall@k

DCG@k

IdealDCG temp: needs to be sorted

Sort ideal_dcg in decreasing relevance order and obtain IdealDCG@k

Compute nDCG@k = DCG@k / IdealDCG@k

Compute Average Precision and Average DCG

Definition at line 190 of file Evaluator.cpp.

5.2.3.5 get_average_dcg()

```
double Evaluator::get_average_dcg ( )
```

Definition at line 343 of file Evaluator.cpp.

5.2.3.6 get_average_ndcg()

```
double Evaluator::get_average_ndcg ( )
```

Definition at line 344 of file Evaluator.cpp.

5.2.3.7 get_average_precision()

```
double Evaluator::get_average_precision ( )
```

Definition at line 341 of file Evaluator.cpp.

5.2.3.8 get_average_recall()

```
double Evaluator::get_average_recall ( )
```

Definition at line 342 of file Evaluator.cpp.

5.2.3.9 get_dcg()

```
double Evaluator::get_dcg (
            uint32_t i )
```

Definition at line 367 of file Evaluator.cpp.

5.2.3.10 get_F1()

```
double Evaluator::get_F1 (
            uint32_t i )
```

Definition at line 360 of file Evaluator.cpp.

5.2.3.11 get_ndcg()

```
double Evaluator::get_ndcg (
            uint32_t i )  [inline]
```

Definition at line 374 of file Evaluator.cpp.

5.2.3.12 get_num_rel()

```
uint32_t Evaluator::get_num_rel ( )
```

Accessors.

Definition at line 338 of file Evaluator.cpp.

5.2.3.13 get_precision()

```
double Evaluator::get_precision (
            uint32_t i )
```

Definition at line 346 of file Evaluator.cpp.

5.2.3.14 get_recall()

```
double Evaluator::get_recall (
            uint32_t i )
```

Definition at line 353 of file Evaluator.cpp.

5.2.3.15 get_true_positives()

```
uint32_t Evaluator::get_true_positives ( )
```

Definition at line 339 of file Evaluator.cpp.

5.2.3.16 insert_relev()

```
void Evaluator::insert_relev (
            char * r,
            uint32_t j )
```

Insert a relevance judgement into the relevs lexicon.

Definition at line 27 of file Evaluator.cpp.

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

- FLAGR/src/Evaluator.h
- FLAGR/src/Evaluator.cpp

5.3 InputData Class Reference

```
#include <InputData.h>
```

Public Member Functions

- InputData ()

  Default Constructor.
- InputData (class InputParams *PARAMS)

  Constructor 2 receives the input parameters as argument.
- ~InputData ()

  Destructor.
- void aggregate ()

  Apply the selected rank aggregation method and construct the output lists of each Aggregator.
- void evaluate ()
- void evaluate_input ()

  Evaluate the input lists of each query.
- void destroy_output_lists ()

  Dstroy the output lists for all quesries.
- void print_header ()

  Print execution information in stdout.
- uint32_t get_num_queries ()

  Accessors.
- class Query * get_query (uint32_t)
- double get_MAP ()
- double get_MNDCG ()
- rank_t get_num_ret ()
- rank_t get_num_rel ()
- rank_t get_num_rel_ret ()
- double get_mean_precision (uint32_t)
- double get_mean_recall (uint32_t)
- double get_mean_F1 (uint32_t)
- double get_mean_dcg (uint32_t)
- double get_mean_ndcg (uint32_t)
- double get_avg_sprho ()
- FILE * get_eval_file ()
- uint32_t compute_avg_list_length ()

  Compute the average list length per query.

5.3.1 Detailed Description

Definition at line 4 of file InputData.h.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 InputData() [1/2]

```
InputData::InputData ( )
```

Default Constructor.

Definition at line 7 of file InputData.cpp.

5.3.2.2 InputData() [2/2]

```
InputData::InputData (
            class InputParams * PARAMS )
```

Constructor 2 receives the input parameters as argument.

Read a single CSV file (all voters and all queries in the same file)

Definition at line 25 of file InputData.cpp.

5.3.2.3 ~InputData()

```
InputData::~InputData ( )
```

Destructor.

Definition at line 77 of file InputData.cpp.

5.3.3 Member Function Documentation

5.3.3.1 aggregate()

```
void InputData::aggregate ( )
```

Apply the selected rank aggregation method and construct the output lists of each Aggregator.

Definition at line 333 of file InputData.cpp.

5.3.3.2 compute_avg_list_length()

```
uint32_t InputData::compute_avg_list_length ( )
```

Compute the average list length per query.

Definition at line 347 of file InputData.cpp.

5.3.3.3 destroy_output_lists()

```
void InputData::destroy_output_lists ( )
```

Dstroy the output lists for all quesries.

Definition at line 115 of file InputData.cpp.

5.3.3.4 evaluate()

```
void InputData::evaluate ( )
```

Evaluate the aggregate lists that have been constructed for each query. This one must be called the aggregate function of each query. Initialize the retrieval effectiveness metrics

Write the header row in the CSV evaluation file

Evaluate each query

Update the accumulators so that we can compute the mean values in the end

Compute the mean values

Create a last row in the CSV evaluation file with the mean values

Definition at line 224 of file InputData.cpp.

5.3.3.5 evaluate_input()

```
void InputData::evaluate_input ( )
```

Evaluate the input lists of each query.

Initialize the retrieval effectiveness metrics

Definition at line 321 of file InputData.cpp.

5.3.3.6 get_avg_sprho()

```
double InputData::get_avg_sprho ( )
```

Definition at line 368 of file InputData.cpp.

5.3.3.7 get_eval_file()

```
FILE * InputData::get_eval_file ( )
```

5.3.3.8 get_MAP()

```
double InputData::get_MAP ( )
```

Definition at line 366 of file InputData.cpp.

5.3.3.9 get_mean_dcg()

```
double InputData::get_mean_dcg (
            uint32_t i )
```

Definition at line 364 of file InputData.cpp.

5.3.3.10 get_mean_F1()

```
double InputData::get_mean_F1 (
            uint32_t i )
```

Definition at line 363 of file InputData.cpp.

5.3.3.11 get_mean_ndcg()

```
double InputData::get_mean_ndcg (
            uint32_t i )
```

Definition at line 365 of file InputData.cpp.

5.3.3.12 get_mean_precision()

```
double InputData::get_mean_precision (
            uint32_t i )
```

Definition at line 361 of file InputData.cpp.

5.3.3.13 get_mean_recall()

```
double InputData::get_mean_recall (
            uint32_t i )
```

Definition at line 362 of file InputData.cpp.

5.3.3.14 get_MNDCG()

```
double InputData::get_MNDCG ( )
```

Definition at line 367 of file InputData.cpp.

5.3.3.15 get_num_queries()

```
uint32_t InputData::get_num_queries ( )
```

Accessors.

Definition at line 359 of file InputData.cpp.

5.3.3.16 get_num_rel()

```
rank_t InputData::get_num_rel ( )
```

Definition at line 370 of file InputData.cpp.

5.3.3.17 get_num_rel_ret()

```
rank_t InputData::get_num_rel_ret ( )
```

Definition at line 371 of file InputData.cpp.

5.3.3.18 get_num_ret()

```
rank_t InputData::get_num_ret ( )
```

Definition at line 369 of file InputData.cpp.

5.3.3.19 get_query()

```
class Query * InputData::get_query (
            uint32_t i )
```

Definition at line 360 of file InputData.cpp.

5.3.3.20 print_header()

```
void InputData::print_header ( )
```

Print execution information in stdout.

Definition at line 149 of file InputData.cpp.

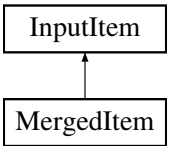
The documentation for this class was generated from the following files:

- FLAGR/src/input/InputData.h
- FLAGR/src/input/InputData.cpp
- FLAGR/src/input/InputDataCSV.cpp
- FLAGR/src/input/InputDataTSV.cpp

5.4 InputItem Class Reference

```
#include <InputItem.h>
```

Inheritance diagram for InputItem:



Public Member Functions

- InputItem ()
  Default Constructor.
- InputItem (char *, rank_t, score_t)
  Constructor 2: overloaded.
- ~InputItem ()
  Destructor.
- void display ()
  Display InputItem Object.
- void set_code (char *)
  Mutators.
- void set_rank (rank_t)
- void set_inscore (score_t)
- char * get_code ()
  Accessors.
- rank_t get_rank ()
- score_t get_inscore ()

Protected Attributes

- char * code
- rank_t rank

  A unique identifier used to identify the common list elements.
- score_t inscore

  The ranking of this item in its input list.

5.4.1 Detailed Description

Definition at line 5 of file InputItem.h.

5.4.2 Constructor & Destructor Documentation

5.4.2.1 InputItem() [1/2]

```
InputItem::InputItem ( )
```

Default Constructor.

Definition at line 4 of file InputItem.cpp.

5.4.2.2 InputItem() [2/2]

```
InputItem::InputItem (
            char * c,
            rank_t r,
            score_t s )
```

Constructor 2: overloaded.

Definition at line 7 of file InputItem.cpp.

5.4.2.3 ~InputItem()

```
InputItem::~InputItem ( )
```

Destructor.

Definition at line 12 of file InputItem.cpp.

5.4.3 Member Function Documentation

5.4.3.1 display()

```
void InputItem::display ( )
```

Display InputItem Object.

Definition at line 25 of file InputItem.cpp.

5.4.3.2 get_code()

```
char * InputItem::get_code ( )
```

Accessors.

Definition at line 35 of file InputItem.cpp.

5.4.3.3 get_inscore()

```
score_t InputItem::get_inscore ( )
```

Definition at line 37 of file InputItem.cpp.

5.4.3.4 get_rank()

```
rank_t InputItem::get_rank ( )
```

Definition at line 36 of file InputItem.cpp.

5.4.3.5 set_code()

```
void InputItem::set_code (
            char * v )
```

Mutators.

Definition at line 30 of file InputItem.cpp.

5.4.3.6 set_inscore()

```
void InputItem::set_inscore (
            score_t v )
```

Definition at line 32 of file InputItem.cpp.

5.4.3.7 set_rank()

```
void InputItem::set_rank (
            rank_t v )
```

Definition at line 31 of file InputItem.cpp.

5.4.4 Member Data Documentation

5.4.4.1 code

```
char* InputItem::code  [protected]
```

Definition at line 7 of file InputItem.h.

5.4.4.2 inscore

```
score_t InputItem::inscore  [protected]
```

The ranking of this item in its input list.

Definition at line 9 of file InputItem.h.

5.4.4.3 rank

```
rank_t InputItem::rank  [protected]
```

A unique identifier used to identify the common list elements.

Definition at line 8 of file InputItem.h.

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

- FLAGR/src/InputItem.h
- FLAGR/src/InputItem.cpp

5.5 InputList Class Reference

`#include <InputList.h>`

Public Member Functions

- InputList ()
  Default Constructor.
- InputList (uint32_t, char *, score_t)
  Constructor 2.
- ~InputList ()
  Destructor.
- void insert_item (char *, rank_t, score_t)
  Insert an item into the list.
- void replace_item (char *, rank_t, score_t)
  Replace an item of the list.
- class InputItem * search_item (char *)
  Search for an item in the list.
- void display ()
  Display the Input List Data.
- void sort_by_score ()
- score_t SpearmanRho (class InputList *)
  Compute the Spearman rho correlation of this list with another input list.
- void set_id (uint32_t)
  Mutators.
- void set_voter_weight (double)
- void set_cutoff (rank_t)
- uint32_t get_id ()
  Accessors.
- class Voter * get_voter ()
- rank_t get_num_items ()
- rank_t get_cutoff ()
- class InputItem * get_item (rank_t)
- score_t get_min_score ()
- score_t get_max_score ()
- score_t get_mean_score ()
- score_t get_std_score ()

5.5.1 Detailed Description

Definition at line 5 of file InputList.h.

5.5.2 Constructor & Destructor Documentation

5.5.2.1 InputList() [1/2]

```
InputList::InputList ( )
```

Default Constructor.

Definition at line 4 of file InputList.cpp.

5.5.2.2 InputList() [2/2]

```
InputList::InputList (
            uint32_t i,
            char * v,
            score_t w )
```

Constructor 2.

Definition at line 14 of file InputList.cpp.

5.5.2.3 ~InputList()

```
InputList::~InputList ( )
```

Destructor.

Definition at line 24 of file InputList.cpp.

5.5.3 Member Function Documentation

5.5.3.1 display()

```
void InputList::display ( )
```

Display the Input List Data.

Definition at line 137 of file InputList.cpp.

5.5.3.2 get_cutoff()

```
rank_t InputList::get_cutoff ( )
```

Definition at line 157 of file InputList.cpp.

5.5.3.3 get_id()

```
uint32_t InputList::get_id ( )
```

Accessors.

Definition at line 153 of file InputList.cpp.

5.5.3.4 get_item()

```
class InputItem * InputList::get_item (
            rank_t i )
```

Definition at line 156 of file InputList.cpp.

5.5.3.5 get_max_score()

```
score_t InputList::get_max_score ( )
```

Definition at line 160 of file InputList.cpp.

5.5.3.6 get_mean_score()

```
score_t InputList::get_mean_score ( )
```

Definition at line 161 of file InputList.cpp.

5.5.3.7 get_min_score()

```
score_t InputList::get_min_score ( )
```

Definition at line 159 of file InputList.cpp.

5.5.3.8 get_num_items()

```
rank_t InputList::get_num_items ( )
```

Definition at line 155 of file InputList.cpp.

5.5.3.9 get_std_score()

```
score_t InputList::get_std_score ( )
```

Definition at line 162 of file InputList.cpp.

5.5.3.10 get_voter()

```
class Voter * InputList::get_voter ( )
```

Definition at line 154 of file InputList.cpp.

5.5.3.11 insert_item()

```
void InputList::insert_item (
            char * code,
            rank_t r,
            score_t s )
```

Insert an item into the list.

Definition at line 44 of file InputList.cpp.

5.5.3.12 replace_item()

```
void InputList::replace_item (
            char * code,
            rank_t r,
            score_t s )
```

Replace an item of the list.

Definition at line 59 of file InputList.cpp.

5.5.3.13 search_item()

```
class InputItem * InputList::search_item (
            char * code )
```

Search for an item in the list.

Definition at line 65 of file InputList.cpp.

5.5.3.14 set_cutoff()

```
void InputList::set_cutoff (
            rank_t v )
```

Definition at line 150 of file InputList.cpp.

5.5.3.15 set_id()

```
void InputList::set_id (
            uint32_t v )
```

Mutators.

Definition at line 148 of file InputList.cpp.

5.5.3.16 set_voter_weight()

```
void InputList::set_voter_weight (
            double v )
```

Definition at line 149 of file InputList.cpp.

5.5.3.17 sort_by_score()

```
void InputList::sort_by_score ( )
```

Sort the list in decreasing order of its element scores. Then, update the item rankings and compute four score statistics: min, max, mean, and std. Set the min/max score values

Compute and set the mean score value

Compute and set the standard deviation of the element scores.

Definition at line 106 of file InputList.cpp.

5.5.3.18 SpearmanRho()

```
score_t InputList::SpearmanRho (
            class InputList * in )
```

Compute the Spearman rho correlation of this list with another input list.

Definition at line 75 of file InputList.cpp.

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

- FLAGR/src/InputList.h
- FLAGR/src/InputList.cpp

5.6 InputParams Class Reference

`#include <InputParams.h>`

Public Member Functions

- InputParams ()
  Default Constructor.
- InputParams (struct UserParams)
  Default Constructor.
- ~InputParams ()
  Destructor.
- void set_output_files (char *)
  Prepare the output files (aggregation output and evaluation report)
- void display ()
  Display members.
- char * get_input_file ()
  Accessors.
- char * get_rels_file ()
- char * get_output_file ()
- char * get_eval_file ()
- char * get_random_string ()
- uint32_t get_aggregation_method ()
- uint32_t get_correlation_method ()
- uint32_t get_weights_normalization ()
- int32_t get_max_iterations ()
- int32_t get_iterations ()
- uint32_t get_max_list_items ()
- rank_t get_eval_points ()
- bool get_list_pruning ()
- bool get_exact ()
- score_t get_convergence_precision ()
- score_t get_alpha ()
- score_t get_beta ()
- score_t get_gamma ()
- score_t get_delta1 ()
- score_t get_delta2 ()
- score_t get_c1 ()
- score_t get_c2 ()
- score_t get_pref_thr ()
- score_t get_veto_thr ()
- score_t get_conc_thr ()
- score_t get_disc_thr ()
- void set_input_file (char *)
  Mutators.
- void set_rels_file (char *)
- void set_output_file (const char *)
- void set_eval_file (const char *)
- void set_random_string (const char *)
- void set_aggregation_method (uint32_t)
- void set_correlation_method (uint32_t)
- void set_weights_normalization (uint32_t)

- void set_max_iterations (int32_t)
- void set_iterations (int32_t)
- void set_max_list_items (uint32_t)
- void set_eval_points (rank_t)
- void set_list_pruning (bool)
- void set_convergence_precision (score_t)
- void set_alpha (score_t)
- void set_beta (score_t)
- void set_gamma (score_t)
- void set_delta1 (score_t)
- void set_delta2 (score_t)
- void set_c1 (score_t)
- void set_c2 (score_t)
- void set_pref_thr (score_t)
- void set_veto_thr (score_t)
- void set_conc_thr (score_t)
- void set_disc_thr (score_t)

5.6.1 Detailed Description

RANK AGGREGATION METHODS (uint32_t aggregation_method) 100: CombSUM with Borda normalization [1] 101: CombSUM with Rank normalization [1] 102: CombSUM with Score normalization [1] 103: CombSUM with Z-Score normalization [1] 104: CombSUM with SimpleBorda normalization 110: CombMNZ with Borda normalization [1] 111: CombMNZ with Rank normalization [1] 112: CombMNZ with Score normalization [1] 113: CombMNZ with Z-Score normalization [1] 114: CombMNZ with SimpleBorda normalization 200: Condorcet Winners Method 201: Copeland Winners Method 300: Outranking Approach [2] 400: Kemeny Optimal Aggregation (Brute Force) 401: Robust Rank Aggregation (RRA) [7]

1. DIBRA @ CombSUM with Borda Normalization [5]
2. DIBRA @ CombSUM with Rank Normalization [5]
3. DIBRA @ CombSUM with Score Normalization [5]
4. DIBRA @ CombSUM with Z-Score Normalization [5]
5. DIBRA @ CombSUM with SimpleBorda Normalization [5]
6. DIBRA @ CombMNZ with Borda Normalization [5]
7. DIBRA @ CombMNZ with Rank Normalization [5]
8. DIBRA @ CombMNZ with Score Normalization [5]
9. DIBRA @ CombMNZ with Z-Score Normalization [5]
10. DIBRA @ CombMNZ with SimpleBorda Normalization [5]
11. DIBRA @ Condorcet Winners [5]
12. DIBRA @ Copeland Winners [5]
13. DIBRA @ Outranking Approach [5]

1. Preference Relations Method [3]
2. Agglomerative Aggregation [4]
3. Markov Chains 1 (MC1) [6]

4. Markov Chains 2 (MC2) [6]
5. Markov Chains 3 (MC3) [6]
6. Markov Chains 4 (MC4) [6]
7. Markov Chains Thurstone (MCT) [9] LIST CORRELATION/DISTANCE MEASURES (uint32_t correlation↩ _method) 1: Spearman's Rho 2: Scaled Footrule Distance 3: Cosine Similarity 4: Local Scaled Footrule Distance 5: Kendall's Tau VOTER WEIGHTS NORMALIZATION METHOD (uint32_t weights_normalization) 1: No normalization (use raw values) 2: Min-Max 3: Z-score 4: Division my max (same as 1)

Definition at line 88 of file InputParams.h.

5.6.2 Constructor & Destructor Documentation

5.6.2.1 InputParams() [1/2]

```
InputParams::InputParams ( )
```

Default Constructor.

Definition at line 4 of file InputParams.cpp.

5.6.2.2 InputParams() [2/2]

```
InputParams::InputParams (
            struct UserParams uParams )
```

Default Constructor.

Definition at line 32 of file InputParams.cpp.

5.6.2.3 ~InputParams()

```
InputParams::~InputParams ( )
```

Destructor.

Definition at line 69 of file InputParams.cpp.

5.6.3 Member Function Documentation

5.6.3.1 display()

```
void InputParams::display ( )
```

Display members.

Definition at line 78 of file InputParams.cpp.

5.6.3.2 get_aggregation_method()

```
uint32_t InputParams::get_aggregation_method ( )
```

Definition at line 169 of file InputParams.cpp.

5.6.3.3 get_alpha()

```
score_t InputParams::get_alpha ( )
```

Definition at line 179 of file InputParams.cpp.

5.6.3.4 get_beta()

```
score_t InputParams::get_beta ( )
```

Definition at line 180 of file InputParams.cpp.

5.6.3.5 get_c1()

```
score_t InputParams::get_c1 ( )
```

Definition at line 184 of file InputParams.cpp.

5.6.3.6 get_c2()

```
score_t InputParams::get_c2 ( )
```

Definition at line 185 of file InputParams.cpp.

5.6.3.7 get_conc_thr()

```
score_t InputParams::get_conc_thr ( )
```

Definition at line 188 of file InputParams.cpp.

5.6.3.8 get_convergence_precision()

```
score_t InputParams::get_convergence_precision ( )
```

Definition at line 178 of file InputParams.cpp.

5.6.3.9 get_correlation_method()

```
uint32_t InputParams::get_correlation_method ( )
```

Definition at line 170 of file InputParams.cpp.

5.6.3.10 get_delta1()

```
score_t InputParams::get_delta1 ( )
```

Definition at line 182 of file InputParams.cpp.

5.6.3.11 get_delta2()

```
score_t InputParams::get_delta2 ( )
```

Definition at line 183 of file InputParams.cpp.

5.6.3.12 get_disc_thr()

```
score_t InputParams::get_disc_thr ( )
```

Definition at line 189 of file InputParams.cpp.

5.6.3.13 get_eval_file()

```
char * InputParams::get_eval_file ( )
```

Definition at line 166 of file InputParams.cpp.

5.6.3.14 get_eval_points()

```
rank_t InputParams::get_eval_points ( )
```

Definition at line 174 of file InputParams.cpp.

5.6.3.15 get_exact()

```
bool InputParams::get_exact ( )
```

Definition at line 176 of file InputParams.cpp.

5.6.3.16 get_gamma()

```
score_t InputParams::get_gamma ( )
```

Definition at line 181 of file InputParams.cpp.

5.6.3.17 get_input_file()

```
char * InputParams::get_input_file ( )
```

Accessors.

Definition at line 163 of file InputParams.cpp.

5.6.3.18 get_iterations()

```
int32_t InputParams::get_iterations ( )
```

5.6.3.19 get_list_pruning()

```
bool InputParams::get_list_pruning ( )
```

Definition at line 175 of file InputParams.cpp.

5.6.3.20 get_max_iterations()

```
int32_t InputParams::get_max_iterations ( )
```

Definition at line 172 of file InputParams.cpp.

5.6.3.21 get_max_list_items()

```
uint32_t InputParams::get_max_list_items ( )
```

Definition at line 173 of file InputParams.cpp.

5.6.3.22 get_output_file()

```
char * InputParams::get_output_file ( )
```

Definition at line 165 of file InputParams.cpp.

5.6.3.23 get_pref_thr()

```
score_t InputParams::get_pref_thr ( )
```

Definition at line 186 of file InputParams.cpp.

5.6.3.24 get_random_string()

```
char * InputParams::get_random_string ( )
```

Definition at line 167 of file InputParams.cpp.

5.6.3.25 get_rels_file()

```
char * InputParams::get_rels_file ( )
```

Definition at line 164 of file InputParams.cpp.

5.6.3.26 get_veto_thr()

```
score_t InputParams::get_veto_thr ( )
```

Definition at line 187 of file InputParams.cpp.

5.6.3.27 get_weights_normalization()

```
uint32_t InputParams::get_weights_normalization ( )
```

Definition at line 171 of file InputParams.cpp.

5.6.3.28 set_aggregation_method()

```
void InputParams::set_aggregation_method (
            uint32_t v )
```

Definition at line 219 of file InputParams.cpp.

5.6.3.29 set_alpha()

```
void InputParams::set_alpha (
            score_t v )
```

Definition at line 228 of file InputParams.cpp.

5.6.3.30 set_beta()

```
void InputParams::set_beta (
            score_t v )
```

Definition at line 229 of file InputParams.cpp.

5.6.3.31 set_c1()

```
void InputParams::set_c1 (
            score_t v )
```

Definition at line 233 of file InputParams.cpp.

5.6.3.32 set_c2()

```
void InputParams::set_c2 (
            score_t v )
```

Definition at line 234 of file InputParams.cpp.

5.6.3.33 set_conc_thr()

```
void InputParams::set_conc_thr (
            score_t v )
```

Definition at line 237 of file InputParams.cpp.

5.6.3.34 set_convergence_precision()

```
void InputParams::set_convergence_precision (
            score_t v )
```

Definition at line 227 of file InputParams.cpp.

5.6.3.35 set_correlation_method()

```
void InputParams::set_correlation_method (
            uint32_t v )
```

Definition at line 220 of file InputParams.cpp.

5.6.3.36 set_delta1()

```
void InputParams::set_delta1 (
            score_t v )
```

Definition at line 231 of file InputParams.cpp.

5.6.3.37 set_delta2()

```
void InputParams::set_delta2 (
            score_t v )
```

Definition at line 232 of file InputParams.cpp.

5.6.3.38 set_disc_thr()

```
void InputParams::set_disc_thr (
            score_t v )
```

Definition at line 238 of file InputParams.cpp.

5.6.3.39 set_eval_file()

```
void InputParams::set_eval_file (
            const char * v )
```

Definition at line 209 of file InputParams.cpp.

5.6.3.40 set_eval_points()

```
void InputParams::set_eval_points (
            rank_t v )
```

Definition at line 224 of file InputParams.cpp.

5.6.3.41 set_gamma()

```
void InputParams::set_gamma (
            score_t v )
```

Definition at line 230 of file InputParams.cpp.

5.6.3.42 set_input_file()

```
void InputParams::set_input_file (
            char * v )
```

Mutators.

// Mutators

Definition at line 194 of file InputParams.cpp.

5.6.3.43 set_iterations()

```
void InputParams::set_iterations (
            int32_t  )
```

5.6.3.44 set_list_pruning()

```
void InputParams::set_list_pruning (
            bool v )
```

Definition at line 225 of file InputParams.cpp.

5.6.3.45 set_max_iterations()

```
void InputParams::set_max_iterations (
            int32_t v )
```

Definition at line 222 of file InputParams.cpp.

5.6.3.46 set_max_list_items()

```
void InputParams::set_max_list_items (
            uint32_t v )
```

Definition at line 223 of file InputParams.cpp.

5.6.3.47 set_output_file()

```
void InputParams::set_output_file (
            const char * v )
```

Definition at line 204 of file InputParams.cpp.

5.6.3.48 set_output_files()

```
void InputParams::set_output_files (
            char * out_dir )
```

Prepare the output files (aggregation output and evaluation report)

Definition at line 137 of file InputParams.cpp.

5.6.3.49 set_pref_thr()

```
void InputParams::set_pref_thr (
            score_t v )
```

Definition at line 235 of file InputParams.cpp.

5.6.3.50 set_random_string()

```
void InputParams::set_random_string (
            const char * v )
```

Definition at line 214 of file InputParams.cpp.

5.6.3.51 set_rels_file()

```
void InputParams::set_rels_file (
            char * v )
```

Definition at line 199 of file InputParams.cpp.

5.6.3.52 set_veto_thr()

```
void InputParams::set_veto_thr (
            score_t v )
```

Definition at line 236 of file InputParams.cpp.

5.6.3.53 set_weights_normalization()

```
void InputParams::set_weights_normalization (
            uint32_t v )
```

Definition at line 221 of file InputParams.cpp.

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

- FLAGR/src/InputParams.h
- FLAGR/src/InputParams.cpp

5.7 max_similarity Struct Reference

Public Attributes

- score_t sim
- int32_t merge_with

5.7.1 Detailed Description

The Agglomerative Aggregation Algorithm of Chatterjee et. al, 2018. Published in: Chatterjee, S., Mukhopadhyay, A., Bhattacharyya, M., "A weighted rank aggregation approach towards crowd opinion analysis", Knowledge-Based Systems, vol. 149, pp. 47-60, 2018. // An assistant structure that stores a MergedList, and a similarity/correlation value. During the Agglomerative Aggregation, we maintain one such record for each input list.

Definition at line 8 of file Agglomerative.cpp.

5.7.2 Member Data Documentation

5.7.2.1 merge_with

```
int32_t max_similarity::merge_with
```

Definition at line 10 of file Agglomerative.cpp.

5.7.2.2 sim

```
score_t max_similarity::sim
```

Definition at line 9 of file Agglomerative.cpp.

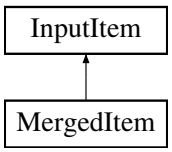
The documentation for this struct was generated from the following file:

- FLAGR/src/ram/Agglomerative.cpp

5.8 MergedItem Class Reference

```
#include <MergedItem.h>
```

Inheritance diagram for MergedItem:



Public Member Functions

- MergedItem ()
  Default Constructor.
- MergedItem (class MergedItem *)
  Constructor 2.
- MergedItem (char *, rank_t, uint32_t, class InputList **)
  Constructor 3.
- ~MergedItem ()
  Destructor.
- void insert_ranking (class InputList *, rank_t, score_t)
  Insert a ranking into the MergedItem.
- void sort_rankings_by_score ()
  Sort the individual item rankings in increasing score order.
- void compute_beta_values ()
  Compute the beta values of the ranking scores.
- void display ()
  Display the MergedItem data.
- void set_final_score (score_t)
  Mutators.
- void set_final_ranking (rank_t)
- void set_next (class MergedItem *)
- score_t get_final_score ()
  Accessors.
- rank_t get_final_ranking ()
- uint32_t get_num_rankings ()
- uint32_t get_num_alloc_rankings ()
- class MergedItem * get_next ()
- class Ranking * get_ranking (uint32_t)

Additional Inherited Members

5.8.1 Detailed Description

Definition at line 4 of file MergedItem.h.

5.8.2 Constructor & Destructor Documentation

5.8.2.1 MergedItem() [1/3]

```
MergedItem::MergedItem ( )
```

Default Constructor.

Definition at line 5 of file MergedItem.cpp.

5.8.2.2 MergedItem() [2/3]

```
MergedItem::MergedItem (
            class MergedItem * in )
```

Constructor 2.

Definition at line 15 of file MergedItem.cpp.

5.8.2.3 MergedItem() [3/3]

```
MergedItem::MergedItem (
            char * c,
            rank_t r,
            uint32_t nal,
            class InputList ** l )
```

Constructor 3.

Definition at line 35 of file MergedItem.cpp.

5.8.2.4 ~MergedItem()

```
MergedItem::~MergedItem ( )
```

Destructor.

Definition at line 49 of file MergedItem.cpp.

5.8.3 Member Function Documentation

5.8.3.1 compute_beta_values()

```
void MergedItem::compute_beta_values ( )
```

Compute the beta values of the ranking scores.

Definition at line 103 of file MergedItem.cpp.

5.8.3.2 display()

```
void MergedItem::display ( )
```

Display the MergedItem data.

Definition at line 71 of file MergedItem.cpp.

5.8.3.3 get_final_ranking()

```
rank_t MergedItem::get_final_ranking ( )
```

Definition at line 123 of file MergedItem.cpp.

5.8.3.4 get_final_score()

```
score_t MergedItem::get_final_score ( )
```

Accessors.

Definition at line 122 of file MergedItem.cpp.

5.8.3.5 get_next()

```
class MergedItem * MergedItem::get_next ( )
```

Definition at line 126 of file MergedItem.cpp.

5.8.3.6 get_num_alloc_rankings()

```
uint32_t MergedItem::get_num_alloc_rankings ( )
```

Definition at line 125 of file MergedItem.cpp.

5.8.3.7 get_num_rankings()

```
uint32_t MergedItem::get_num_rankings ( )
```

Definition at line 124 of file MergedItem.cpp.

5.8.3.8 get_ranking()

```
class Ranking * MergedItem::get_ranking (
            uint32_t i )
```

Definition at line 127 of file MergedItem.cpp.

5.8.3.9 insert_ranking()

```
void MergedItem::insert_ranking (
            class InputList * l,
            rank_t r,
            score_t s )
```

Insert a ranking into the MergedItem.

Definition at line 62 of file MergedItem.cpp.

5.8.3.10 set_final_ranking()

```
void MergedItem::set_final_ranking (
            rank_t v )
```

Definition at line 118 of file MergedItem.cpp.

5.8.3.11 set_final_score()

```
void MergedItem::set_final_score (
            score_t v )
```

Mutators.

Definition at line 117 of file MergedItem.cpp.

5.8.3.12 set_next()

```
void MergedItem::set_next (
            class MergedItem * v )
```

Definition at line 119 of file MergedItem.cpp.

5.8.3.13 sort_rankings_by_score()

```
void MergedItem::sort_rankings_by_score ( )
```

Sort the individual item rankings in increasing score order.

Definition at line 86 of file MergedItem.cpp.

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

- FLAGR/src/MergedItem.h
- FLAGR/src/MergedItem.cpp

5.9 MergedItemPair Class Reference

```
#include <MergedItemPair.h>
```

Public Member Functions

- MergedItemPair ()
- MergedItemPair (class MergedItem ∗, class MergedItem ∗)

  Constructor 1.
- ∼MergedItemPair ()

  Destructor.
- void compute_a_majority_opinion (score_t a, score_t b, uint32_t N)

  Compute the a-majority opinion.
- void compute_a_majority_opinion_debug (score_t a, score_t b, uint32_t N)

  Compute a-majority opinion (debug mode)
- void compute_weight ()
- void display (uint32_t)

  Display the ItemPair.
- class MergedItem ∗ get_item1 ()

  Accessors.
- class MergedItem ∗ get_item2 ()
- score_t get_score ()
- void set_item1 (class MergedItem ∗)

  Mutators.
- void set_item2 (class MergedItem ∗)
- void set_score (score_t)

5.9.1 Detailed Description

Definition at line 5 of file MergedItemPair.h.

5.9.2 Constructor & Destructor Documentation

5.9.2.1 MergedItemPair() [1/2]

```
MergedItemPair::MergedItemPair ( )
```

Used in the Preference Relations method of [3] Default Constructor

Definition at line 5 of file MergedItemPair.cpp.

5.9.2.2 MergedItemPair() [2/2]

```
MergedItemPair::MergedItemPair (
            class MergedItem * i1,
            class MergedItem * i2 )
```

Constructor 1.

Definition at line 8 of file MergedItemPair.cpp.

5.9.2.3 ~MergedItemPair()

```
MergedItemPair::~MergedItemPair ( )
```

Destructor.

Definition at line 13 of file MergedItemPair.cpp.

5.9.3 Member Function Documentation

5.9.3.1 compute_a_majority_opinion()

```
void MergedItemPair::compute_a_majority_opinion (
            score_t a,
            score_t b,
            uint32_t N )
```

Compute the a-majority opinion.

Find the number of lists that agree (=n0) or disagree(=n1) with the ranking (item1, item2).

Update the disagreement scores of each list according to Eq. 4 of [3].

Eq. 2 is satisfied for x = 0

Definition at line 17 of file MergedItemPair.cpp.

5.9.3.2 compute_a_majority_opinion_debug()

```
void MergedItemPair::compute_a_majority_opinion_debug (
            score_t a,
            score_t b,
            uint32_t N )
```

Compute a-majority opinion (debug mode)

Find the number of lists that agree (=n0) or disagree(=n1) with the ranking (item1, item2).

Update the disagreement scores of each list according to Eq. 4 of [3].

Eq. 2 is satisfied for x = 0

Definition at line 70 of file MergedItemPair.cpp.

5.9.3.3 compute_weight()

```
void MergedItemPair::compute_weight ( )
```

The MergedItemPair is treated as an edge in the aggregate graph. This function computes the weight of the edge. The weight of this directed edge represents the total weighted votes in favor of the preference relation item2 is better than item1 (i.e. r2 $<$ r1).

Definition at line 143 of file MergedItemPair.cpp.

5.9.3.4 display()

```
void MergedItemPair::display (
            uint32_t t )
```

Display the ItemPair.

Definition at line 161 of file MergedItemPair.cpp.

5.9.3.5 get_item1()

```
class MergedItem * MergedItemPair::get_item1 ( )
```

Accessors.

Definition at line 174 of file MergedItemPair.cpp.

5.9.3.6 get_item2()

```
class MergedItem * MergedItemPair::get_item2 ( )
```

Definition at line 175 of file MergedItemPair.cpp.

5.9.3.7 get_score()

```
score_t MergedItemPair::get_score ( )
```

Definition at line 176 of file MergedItemPair.cpp.

5.9.3.8 set_item1()

```
void MergedItemPair::set_item1 (
            class MergedItem * v )
```

Mutators.

Definition at line 179 of file MergedItemPair.cpp.

5.9.3.9 set_item2()

```
void MergedItemPair::set_item2 (
            class MergedItem * v )
```

Definition at line 180 of file MergedItemPair.cpp.

5.9.3.10 set_score()

```
void MergedItemPair::set_score (
            score_t  )
```

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

- FLAGR/src/ram/tools/MergedItemPair.h
- FLAGR/src/ram/tools/MergedItemPair.cpp

5.10 MergedList Class Reference

`#include <MergedList.h>`

Public Member Functions

- MergedList ()
  Constructor 1: default.
- MergedList (class InputList **, uint32_t, uint32_t)
- MergedList (uint32_t, uint32_t)
  Constructor 2: overloaded.
- ~MergedList ()
  Destructor.
- void insert (class InputItem *, uint32_t, class InputList **)
  Insert an element into the hash table.
- void insert_merge (class MergedItem *, score_t)
- void convert_to_array ()
  "Copy" the MergedList's hash table elementd to the internal item_list

- void display ()

  Display the items of the MergedList object (hash_table)
- void display_list ()

  Display the items of the MergedList object (item_list)
- void write_to_CSV (char *, class InputParams *)

  Display the items of the MergedList object (item_list)
- void update_weight (char *, score_t)

  Find an element into the hash table and update its weight.
- void reset_scores ()

  Reset the scores (set to equal to 0) of the merged list elements.
- void reset_weights ()
- void rebuild (class InputList **)

  Rebuild the Merged list from the input lists.
- void clear_contents ()

  Clear all the contents of the Merged List and free all resources - Equivalent to destructor.
- void merge_with (class MergedList *, class InputParams *)

  Merge two temporary aggregate lists: used in the Agglomerative Aggregation algorithm [4].
- rank_t get_item_rank (char *)

  Search for an item and return its rank.
- void CombSUM (class InputList **, class SimpleScoreStats *, class InputParams *)

  Rank Aggregation Methods.
- void CombMNZ (class InputList **, class SimpleScoreStats *, class InputParams *)
- void CondorcetWinners (class InputList **, class SimpleScoreStats *, class InputParams *)
- void CopelandWinners (class InputList **, class SimpleScoreStats *, class InputParams *)
- void Outranking (class InputList **, class SimpleScoreStats *, class InputParams *)
- void KemenyOptimal (class InputList **, class SimpleScoreStats *, class InputParams *)
- void RobustRA (class InputList **, class SimpleScoreStats *, class InputParams *)

  This implementation imitates the one of the RobustRankAggreg package of R.
- void MC (class InputList **, class SimpleScoreStats *, class InputParams *)
- class Voter ** DIBRA (class InputList **, class SimpleScoreStats *, class InputParams *)
- void PrefRel (class InputList **, class SimpleScoreStats *, class InputParams *)
- class MergedList * Agglomerative (class InputList **, class SimpleScoreStats *, class InputParams *)
- void CustomMethod1 (class InputList **, class SimpleScoreStats *, class InputParams *)

  Custom Algorithm declarations.
- void CustomMethod2 (class InputList **, class SimpleScoreStats *, class InputParams *)

  Another custom rank aggregation method.
- double SpearmanRho (class InputList *)

  Rank Correlation Methods.
- double SpearmanRho (class MergedList *)

  Compute the Spearman's rho correlation measure between this MergedList and another one.
- double ScaledFootruleDistance (class MergedList *)

  Compute the Spearman's Footrule distance between this MergedList and another one.
- double ScaledFootruleDistance (uint32_t, class InputList *)

  Scaled Footrule Distance.
- double LocalScaledFootruleDistance (uint32_t, class InputList *)

  Scaled Footrule Distance.
- double CosineSimilarity (uint32_t, class InputList *)
- double KendallsTau (uint32_t, class InputList *)
- rank_t get_num_items ()

  Getters.
- class MergedItem * get_item (uint32_t)
- class MergedItem ** get_item_list ()
- score_t get_weight ()
- void set_weight (score_t)

  Setters.

5.10.1 Detailed Description

Definition at line 5 of file MergedList.h.

5.10.2 Constructor & Destructor Documentation

5.10.2.1 MergedList() [1/3]

```
MergedList::MergedList ( )
```

Constructor 1: default.

Definition at line 18 of file MergedList.cpp.

5.10.2.2 MergedList() [2/3]

```
MergedList::MergedList (
            class InputList ** inlists,
            uint32_t nlists,
            uint32_t m )
```

Constructor 3: overloaded - Create an aggregate list from an input list. Used in Agglomerative Aggregation algorithm of [4] Create a temporary aggregate list from an InputList -> essentially we are converting an InputList into a MergedList. These MergedLists will be progressively merged later in an Agglomerative fashion.

Chatterjee et al. 2018, Eq. 7

Definition at line 53 of file MergedList.cpp.

5.10.2.3 MergedList() [3/3]

```
MergedList::MergedList (
            uint32_t size,
            uint32_t n )
```

Constructor 2: overloaded.

Definition at line 30 of file MergedList.cpp.

5.10.2.4 ~MergedList()

```
MergedList::~MergedList ( )
```

Destructor.

Definition at line 101 of file MergedList.cpp.

5.10.3 Member Function Documentation

5.10.3.1 Agglomerative()

```
class MergedList * MergedList::Agglomerative (
            class InputList ** inlists,
            class SimpleScoreStats * s,
            class InputParams * prms )
```

Which list is the most similar to another one? One record per each input list

Create one temporary aggregate per input list - this will help us in the progressive list merging

Compute the initial list similarities/correlations between the lists

Start merging the lists

Find the pair of the most similar lists

Most similar list

The list that was merged with is deleted (as it is no longer useful)

The list similarities are recomputed for the next iteration

Update the max similarities list

Deallocate all the remaining resources

Definition at line 249 of file Agglomerative.cpp.

5.10.3.2 clear_contents()

```
void MergedList::clear_contents ( )
```

Clear all the contents of the Merged List and free all resources - Equivalent to destructor.

Definition at line 236 of file MergedList.cpp.

5.10.3.3 CombMNZ()

```
void MergedList::CombMNZ (
            class InputList ** inlists,
            class SimpleScoreStats * s,
            class InputParams * prms )
```

The CombMNZ family of linear combination methods. Details about the 5 implemented variants are published in the following paper: Renda E., Straccia U., "Web metasearch: rank vs. score based rank aggregation methods", In Proceedings of the 2003 ACM symposium on Applied computing, pp. 841-846, 2003. // For each element in MergedList, compute the score w.r.t to the selected normalization method

Voter weights normalization

Min-max normalization of voter weights

Z-normalization of voter weights

Division of the voter weights by the maximum voter score

Compute the element scores. Case A: The element has been ranked in list l

Borda normalization: Eq: 4 (first branch) of [1]

Rank normalization: Eq: 3 of [1]

Score normalization: Eq: 1 of [1]

Z-Score normalization: Eq: 2 of [1]

Simple Borda normalization

Case B: The element has NOT been ranked in list l

Borda normalization: Eq: 4 (second branch) of [1]

Rank/Score/Z-Score/SimpleBorda normalization: No scores are assigned to non-ranked elements [1]

Definition at line 7 of file CombMNZ.cpp.

5.10.3.4 CombSUM()

```
void MergedList::CombSUM (
            class InputList ** inlists,
            class SimpleScoreStats * s,
            class InputParams * prms )
```

Rank Aggregation Methods.

The CombSUM family of linear combination methods. Details about the 5 implemented variants are published in the following paper: Renda E., Straccia U., "Web metasearch: rank vs. score based rank aggregation methods", In Proceedings of the 2003 ACM symposium on Applied computing, pp. 841-846, 2003. // For each element in MergedList, compute the score w.r.t to the selected normalization method

Voter weights normalization

Min-max normalization of voter weights

Z-normalization of voter weights

Division of the voter weights by the maximum voter score

Compute the element scores. Case A: The element has been ranked in list l

Borda normalization: Eq: 4 (first branch) of [1]

Rank normalization: Eq: 3 of [1]

Score normalization: Eq: 1 of [1]

Z-Score normalization: Eq: 2 of [1]

Simple Borda normalization

Case B: The element has NOT been ranked in list l

Borda normalization: Eq: 4 (second branch) of [1]

Rank/Score/Z-Score/SimpleBorda normalization: No scores are assigned to non-ranked elements [1]

Definition at line 7 of file CombSUM.cpp.

5.10.3.5 CondorcetWinners()

```
void MergedList::CondorcetWinners (
            class InputList ** inlists,
            class SimpleScoreStats * s,
            class InputParams * prms )
```

The Condorcet Winners method is based on the majority criterion. // Min-max normalization of voter weights

Z normalization of voter weights

Division of the voter weights by the maximum voter score

Definition at line 4 of file CondorcetWinners.cpp.

5.10.3.6 convert_to_array()

```
void MergedList::convert_to_array ( )
```

"Copy" the MergedList's hash table elementd to the internal item_list

Definition at line 197 of file MergedList.cpp.

5.10.3.7 CopelandWinners()

```
void MergedList::CopelandWinners (
            class InputList ** inlists,
            class SimpleScoreStats * s,
            class InputParams * prms )
```

The Copeland Winners method is based on the majority criterion. //
Min-max normalization of voter weights

Z normalization of voter weights

Division of the voter weights by the maximum voter score

Definition at line 4 of file CopelandWinners.cpp.

5.10.3.8 CosineSimilarity()

```
double MergedList::CosineSimilarity (
            uint32_t z,
            class InputList * in )
```

2

BEST

2

Cosine Similarity

Jaccard Index

Sorensen-Dice coefficient

Definition at line 340 of file MergedList.cpp.

5.10.3.9 CustomMethod1()

```
void MergedList::CustomMethod1 (
            class InputList ** inlists,
            class SimpleScoreStats * s,
            class InputParams * prms )
```

Custom Algorithm declarations.

Custom rank aggregation method.

Custom methods implementation Must be declared as public members of MergedList in MergedList.h // Your implementation here

Example iteration through list elements

Get the individual rankings and scores of q in the input lists

Sort the list elements in decreasing score order

or you may want to sort the scores in increasing order qsort(this->item_list, this->num_nodes, sizeof(class MergedItem *), &MergedList::cmp_score_asc);

Definition at line 6 of file CustomMethods.cpp.

5.10.3.10 CustomMethod2()

```
void MergedList::CustomMethod2 (
            class InputList ** inlists,
            class SimpleScoreStats * s,
            class InputParams * prms )
```

Another custom rank aggregation method.

Your implementation here

Sort the list elements in decreasing score order

or you may want to sort the scores in increasing order qsort(this->item_list, this->num_nodes, sizeof(class MergedItem *), &MergedList::cmp_score_asc);

Definition at line 31 of file CustomMethods.cpp.

5.10.3.11 DIBRA()

```
class Voter ** MergedList::DIBRA (
            class InputList ** inlists,
            class SimpleScoreStats * s,
            class InputParams * prms )
```

The Iterative Distance-Based Unsupervised Algorithm (DIBRA) of Akritidis et. al, 2022. Published in: Akritidis L., Fevgas A., Bozanis P., Manolopoulos Y., "An Unsupervised Distance-Based Model for Weighted Rank Aggregation with List Pruning", Expert Systems with Applications, vol. 202, pp. 117435, 2022. /// Reset the weights of all voters

Stop the execution when convergence is achieved

Set the scores of all elements to zero

Execute the baseline method (by taking into consideration the current voter weights)

Initialize the statistics for the weights and the distances

Compute the similarity of each input list with the produced MergedList

Statistics for normalizing the distances - Generally unneeded, because normalization takes place within the distance function itself.

Based on the computed distances of the previous loop, compute the new weights of voters

Compute the new weight of the voter

Exponential

Statistics for normalizing the weights

Set the new voter weights

// Apply the list pruning post-processing step ///

Definition at line 8 of file DIBRA.cpp.

5.10.3.12 display()

```
void MergedList::display ( )
```

Display the items of the MergedList object (hash_table)

Definition at line 161 of file MergedList.cpp.

5.10.3.13 display_list()

```
void MergedList::display_list ( )
```

Display the items of the MergedList object (item_list)

Definition at line 175 of file MergedList.cpp.

5.10.3.14 get_item()

```
class MergedItem * MergedList::get_item (
            uint32_t i )
```

Definition at line 567 of file MergedList.cpp.

5.10.3.15 get_item_list()

```
class MergedItem ** MergedList::get_item_list ( )
```

Definition at line 568 of file MergedList.cpp.

5.10.3.16 get_item_rank()

```
rank_t MergedList::get_item_rank (
            char * c )
```

Search for an item and return its rank.

Definition at line 220 of file MergedList.cpp.

5.10.3.17 get_num_items()

```
rank_t MergedList::get_num_items ( )
```

Getters.

Accessors.

Definition at line 566 of file MergedList.cpp.

5.10.3.18 get_weight()

```
score_t MergedList::get_weight ( )
```

Definition at line 569 of file MergedList.cpp.

5.10.3.19 insert()

```
void MergedList::insert (
            class InputItem * n,
            uint32_t x,
            class InputList ** l )
```

Insert an element into the hash table.

Find the hash value of the input term

Now search in the hash table to check whether this term exists or not

Traverse the linked list that represents the chain.

Return and exit

Create a new record and re-assign the linked list's head

Reassign the chain's head

Definition at line 106 of file MergedList.cpp.

5.10.3.20 insert_merge()

```
void MergedList::insert_merge (
            class MergedItem * item,
            score_t list_weight )
```

Insert/Copy an item from a MergedList to another MergedList: used in the Agglomerative Aggregation algorithm of [4]. Search in the hash table to check whether this item exists or not

Traverse the linked list that represents the chain

The item exists in this MergedList. Insert the new ranking and update its score

Chatterjee et al. 2018, Eq. 8

The item was not found in this MergedList: create a new item record

Reassign the chain's head

Definition at line 15 of file Agglomerative.cpp.

5.10.3.21 KemenyOptimal()

```
void MergedList::KemenyOptimal (
            class InputList ** inlists,
            class SimpleScoreStats * s,
            class InputParams * prms )
```

Definition at line 57 of file KemenyOptimal.cpp.

5.10.3.22 KendallsTau()

```
double MergedList::KendallsTau (
            uint32_t z,
            class InputList * in )
```

Definition at line 315 of file MergedList.cpp.

5.10.3.23 LocalScaledFootruleDistance()

```
double MergedList::LocalScaledFootruleDistance (
            uint32_t z,
            class InputList * in )
```

Scaled Footrule Distance.

Definition at line 424 of file MergedList.cpp.

5.10.3.24 MC()

```
void MergedList::MC (
            class InputList ** inlists,
            class SimpleScoreStats * s,
            class InputParams * prms )
```

The Markov Chains methods of Dwork et. al, 2001. Published in: Dwork C., Kumar R., Naor M., Sivakumar D., "Rank Aggregation Methods for the Web", In Proceedings of the 10th International Conference on World Wide Web, pp. 613-622, 2001. //

Definition at line 6 of file MC.cpp.

5.10.3.25 merge_with()

```
void MergedList::merge_with (
            class MergedList * inlist,
            class InputParams * params )
```

Merge two temporary aggregate lists: used in the Agglomerative Aggregation algorithm [4].

We delete the array of items of this MergedList (we only delete the array, NOT the hast table!)

Traverse the incoming MergedList inlist: Insert each element of inlist into this MergedList

Compute the weight of the updated MergedList from its parents (Ref [4], Eq. 9).

Convert the hash table to an array, so that we can sort by score

Definition at line 70 of file Agglomerative.cpp.

5.10.3.26 Outranking()

```
void MergedList::Outranking (
            class InputList ** inlists,
            class SimpleScoreStats * s,
            class InputParams * prms )
```

The Outranking Approach of Farah and Vanderpooten, 2007. Published in: Farah, M., Vanderpooten, D., "An outranking approach for rank aggregation in information retrieval", In Proceedings of the 30th ACM Conference on Research and Development in Information Retrieval, pp. 591-598, 2007. /// Min-max normalization of voter weights

Z normalization of voter weights

Division of the voter weights by the maximum voter score

Compute the final concordance/discordance score

Definition at line 7 of file OutrankingApproach.cpp.

5.10.3.27 PrefRel()

```
void MergedList::PrefRel (
            class InputList ** inlists,
            class SimpleScoreStats * s,
            class InputParams * prms )
```

The Preference Relations method of Desarkar et. al 2016: Desarkar, M. S., Sarkar, S., Mitra, P., "Preference relations based unsupervised rank aggregation for metasearch", Expert Systems with Applications, vol. 49, pp. 86-98, 2016. /// Form all the item pairs and compute amajority opinions and list disagreement scores

Compute the list weights from their disagreement scores (eq. 5)

Create the edges of the aggregate graph: 2 edges per MergedItem

Sort the edges by their target (right) node

Find the in-degrees of the graph nodes by scanning the edges

Definition at line 8 of file PrefRel.cpp.

5.10.3.28 rebuild()

```
void MergedList::rebuild (
            class InputList ** inlists )
```

Rebuild the Merged list from the input lists.

Definition at line 262 of file MergedList.cpp.

5.10.3.29 reset_scores()

```
void MergedList::reset_scores ( )
```

Reset the scores (set to equal to 0) of the merged list elements.

Definition at line 213 of file MergedList.cpp.

5.10.3.30 reset_weights()

```
void MergedList::reset_weights ( )
```

5.10.3.31 RobustRA()

```
void MergedList::RobustRA (
            class InputList ** inlists,
            class SimpleScoreStats * s,
            class InputParams * prms )
```

This implementation imitates the one of the RobustRankAggreg package of R.

The Robust Rank Aggregation Algorithm of Kolde et. al 2012: Kolde R., Laur S., Adler P., Vilo J., "Robust rank aggregation for gene list integration and meta-analysis", Bioinformatics, vol. 28, no. 4, pp. 573–580, 2012. // Equivalent to the rankMatrix function of RobustRank← Aggreg

Find the p-values of the prevoiusly computed scores, assuming that the elements are distributed with beta distribution .

Sort the rankings of each element

Obtain the probabilites with the incomplete beta probability distribution function See Algorithm ASA063 of [8]

Compute the rho values: corrected p-values Exact correction: We apply the inverted incomplete beta distribution function on the obtained p-values and then, we apply the Stuart-Ares method

Vector for the exact p-value correction

Helper buffer

Helper buffer2

Approximate correction: Just get the minimum p-value that was obtained from the incomplete beta distribution function

Sort the list elements in increasing score order

Definition at line 7 of file RobustRA.cpp.

5.10.3.32 ScaledFootruleDistance() [1/2]

```
double MergedList::ScaledFootruleDistance (
            class MergedList * inlist )
```

Compute the Spearman's Footrule distance between this MergedList and another one.

Definition at line 128 of file Agglomerative.cpp.

5.10.3.33 ScaledFootruleDistance() [2/2]

```
double MergedList::ScaledFootruleDistance (
            uint32_t z,
            class InputList * in )
```

Scaled Footrule Distance.

Definition at line 399 of file MergedList.cpp.

5.10.3.34 set_weight()

```
void MergedList::set_weight (
            score_t v )
```

Setters.

Mutators.

Definition at line 572 of file MergedList.cpp.

5.10.3.35 SpearmanRho() [1/2]

```
double MergedList::SpearmanRho (
            class InputList * in )
```

Rank Correlation Methods.

// RANK CORRELATION DISTANCE METHODS // Between the MergedList and an Input List// //

Definition at line 291 of file MergedList.cpp.

5.10.3.36 SpearmanRho() [2/2]

```
double MergedList::SpearmanRho (
            class MergedList * inlist )
```

Compute the Spearman's rho correlation measure between this MergedList and another one.

Definition at line 100 of file Agglomerative.cpp.

5.10.3.37 update_weight()

```
void MergedList::update_weight (
            char * code,
            score_t w )
```

Find an element into the hash table and update its weight.

Find the hash value of the input term

Now search in the hash table to check whether this term exists or not

Traverse the linked list that represents the chain.

Return and exit

Definition at line 139 of file MergedList.cpp.

5.10.3.38 write_to_CSV()

```
void MergedList::write_to_CSV (
            char * topic,
            class InputParams * params )
```

Display the items of the MergedList object (item_list)

Definition at line 184 of file MergedList.cpp.

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

- FLAGR/src/MergedList.h
- FLAGR/src/MergedList.cpp
- FLAGR/src/ram/Agglomerative.cpp
- FLAGR/src/ram/CombMNZ.cpp
- FLAGR/src/ram/CombSUM.cpp
- FLAGR/src/ram/CondorcetWinners.cpp
- FLAGR/src/ram/CopelandWinners.cpp
- FLAGR/src/ram/CustomMethods.cpp
- FLAGR/src/ram/DIBRA.cpp
- FLAGR/src/ram/KemenyOptimal.cpp
- FLAGR/src/ram/MC.cpp
- FLAGR/src/ram/OutrankingApproach.cpp
- FLAGR/src/ram/PrefRel.cpp
- FLAGR/src/ram/RobustRA.cpp

5.11 Query Class Reference

`#include <Query.h>`

Public Member Functions

- Query (uint32_t)
- ~Query ()

  Destructor.
- class InputList * create_list (char *, double)

  Create a new input list for an aggregator.
- void aggregate (class InputParams *params)

  Apply the rank aggregation method and construct the final output list.
- void insert_relev (char *, uint32_t)

  Apply the rank aggregation method and construct the final output list.
- void display ()

  Display the query properties and input lists.
- void display_relevs ()

  Display the query properties and input lists.
- void evaluate (rank_t, FILE *)

  Evaluate the output list of the query by using the input relevance judgments.
- void evaluate_input ()

  Evaluate all the input lists of the query by using the input relevance judgments.
- void destroy_output_list ()

  Destroy the aggregate list.
- double evaluate_experts_list ()
- void init_weights ()

  Set the initial weights of voters.
- uint32_t get_num_items ()

  Accessors.
- rank_t get_num_rel ()
- rank_t get_num_rel_ret ()
- uint32_t get_num_input_lists ()
- char * get_topic ()
- double get_average_precision ()
- double get_average_recall ()
- double get_average_dcg ()
- double get_average_ndcg ()
- double get_precision (uint32_t)
- double get_recall (uint32_t)
- double get_F1 (uint32_t)
- double get_dcg (uint32_t)
- double get_ndcg (uint32_t)
- class InputList * get_input_list (uint32_t)

  Accessors.
- void set_topic (char *)

  Mutators.

5.11.1 Detailed Description

Definition at line 5 of file Query.h.

5.11.2 Constructor & Destructor Documentation

5.11.2.1 Query()

```
Query::Query (
            uint32_t sw )
```

Constructor 1: If sw == Initialize the evaluator by using the relevance judgments

Definition at line 5 of file Query.cpp.

5.11.2.2 ~Query()

```
Query::~Query ( )
```

Destructor.

Definition at line 18 of file Query.cpp.

5.11.3 Member Function Documentation

5.11.3.1 aggregate()

```
void Query::aggregate (
            class InputParams * params )
```

Apply the rank aggregation method and construct the final output list.

Definition at line 54 of file Query.cpp.

5.11.3.2 create_list()

```
class InputList * Query::create_list (
            char * v,
            double w )
```

Create a new input list for an aggregator.

Definition at line 49 of file Query.cpp.

5.11.3.3 destroy_output_list()

```
void Query::destroy_output_list ( )
```

Destroy the aggregate list.

Definition at line 59 of file Query.cpp.

5.11.3.4 display()

```
void Query::display ( )
```

Display the query properties and input lists.

Definition at line 133 of file Query.cpp.

5.11.3.5 display_relevs()

```
void Query::display_relevs ( )
```

Display the query properties and input lists.

Definition at line 139 of file Query.cpp.

5.11.3.6 evaluate()

```
void Query::evaluate (
            rank_t ev_pts,
            FILE * e_file )
```

Evaluate the output list of the query by using the input relevance judgments.

Definition at line 74 of file Query.cpp.

5.11.3.7 evaluate_experts_list()

```
double Query::evaluate_experts_list ( )
```

Definition at line 78 of file Query.cpp.

5.11.3.8 evaluate_input()

```
void Query::evaluate_input ( )
```

Evaluate all the input lists of the query by using the input relevance judgments.

Definition at line 109 of file Query.cpp.

5.11.3.9 get_average_dcg()

```
double Query::get_average_dcg ( )
```

Definition at line 153 of file Query.cpp.

5.11.3.10 get_average_ndcg()

```
double Query::get_average_ndcg ( )
```

Definition at line 154 of file Query.cpp.

5.11.3.11 get_average_precision()

```
double Query::get_average_precision ( )
```

Definition at line 151 of file Query.cpp.

5.11.3.12 get_average_recall()

```
double Query::get_average_recall ( )
```

Definition at line 152 of file Query.cpp.

5.11.3.13 get_dcg()

```
double Query::get_dcg (
            uint32_t i )
```

Definition at line 158 of file Query.cpp.

5.11.3.14 get_F1()

```
double Query::get_F1 (
            uint32_t i )
```

Definition at line 157 of file Query.cpp.

5.11.3.15 get_input_list()

```
class InputList * Query::get_input_list (
            uint32_t i )
```

Accessors.

Definition at line 145 of file Query.cpp.

5.11.3.16 get_ndcg()

```
double Query::get_ndcg (
            uint32_t i )
```

Definition at line 159 of file Query.cpp.

5.11.3.17 get_num_input_lists()

```
uint32_t Query::get_num_input_lists ( )
```

Definition at line 150 of file Query.cpp.

5.11.3.18 get_num_items()

```
uint32_t Query::get_num_items ( )
```

Accessors.

Definition at line 147 of file Query.cpp.

5.11.3.19 get_num_rel()

```
rank_t Query::get_num_rel ( )
```

Definition at line 148 of file Query.cpp.

5.11.3.20 get_num_rel_ret()

```
rank_t Query::get_num_rel_ret ( )
```

Definition at line 149 of file Query.cpp.

5.11.3.21 get_precision()

```
double Query::get_precision (
            uint32_t i )
```

Definition at line 155 of file Query.cpp.

5.11.3.22 get_recall()

```
double Query::get_recall (
            uint32_t i )
```

Definition at line 156 of file Query.cpp.

5.11.3.23 get_topic()

```
char * Query::get_topic ( )
```

Definition at line 146 of file Query.cpp.

5.11.3.24 init_weights()

```
void Query::init_weights ( )
```

Set the initial weights of voters.

Definition at line 128 of file Query.cpp.

5.11.3.25 insert_relev()

```
void Query::insert_relev (
            char * v,
            uint32_t j )
```

Apply the rank aggregation method and construct the final output list.

Definition at line 69 of file Query.cpp.

5.11.3.26 set_topic()

```
void Query::set_topic (
            char * v )
```

Mutators.

Definition at line 162 of file Query.cpp.

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

- FLAGR/src/Query.h
- FLAGR/src/Query.cpp

5.12 Ranking Class Reference

```
#include <Ranking.h>
```

Public Member Functions

- Ranking (class InputList *, rank_t, score_t)
  The score of an item in the input list.
- ~Ranking ()
  Destructor: nothing to destroy.
- void display ()
  Display Ranking contents.
- void set_input_list (class InputList *)
  Mutators.
- void set_rank (rank_t)
- void set_score (score_t)
- class InputList * get_input_list ()
  Accessors.
- rank_t get_rank ()
- score_t get_score ()

5.12.1 Detailed Description

This structure describes the ranking and the score of an element in an input list; The input_list pointer points to the original input list object.

Definition at line 7 of file Ranking.h.

5.12.2 Constructor & Destructor Documentation

5.12.2.1 Ranking()

```
Ranking::Ranking (
            class InputList * l,
            rank_t r,
            score_t s )
```

The score of an item in the input list.

Constructor 1.

Definition at line 4 of file Ranking.cpp.

5.12.2.2 ~Ranking()

```
Ranking::~Ranking ( )
```

Destructor: nothing to destroy.

Definition at line 10 of file Ranking.cpp.

5.12.3 Member Function Documentation

5.12.3.1 display()

```
void Ranking::display ( )
```

Display Ranking contents.

Definition at line 14 of file Ranking.cpp.

5.12.3.2 get_input_list()

```
class InputList * Ranking::get_input_list ( )
```

Accessors.

Definition at line 25 of file Ranking.cpp.

5.12.3.3 get_rank()

```
rank_t Ranking::get_rank ( )
```

Definition at line 26 of file Ranking.cpp.

5.12.3.4 get_score()

```
score_t Ranking::get_score ( )
```

Definition at line 27 of file Ranking.cpp.

5.12.3.5 set_input_list()

```
void Ranking::set_input_list (
            class InputList * v )
```

Mutators.

Definition at line 20 of file Ranking.cpp.

5.12.3.6 set_rank()

```
void Ranking::set_rank (
            rank_t v )
```

Definition at line 21 of file Ranking.cpp.

5.12.3.7 set_score()

```
void Ranking::set_score (
            score_t v )
```

Definition at line 22 of file Ranking.cpp.

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

- FLAGR/src/Ranking.h
- FLAGR/src/Ranking.cpp

5.13 Rel Class Reference

```
#include <Rel.h>
```

Public Member Functions

- Rel ()
  Constructor 1 :default.
- Rel (char *, uint32_t)
  Constructor 2.
- ~Rel ()
  Destructor.
- void display ()
- void set_code (char *)
  Mutators.
- void set_judgment (uint32_t)
  Mutators.
- void set_next (class Rel *)
- char * get_code ()
  Accessors.
- uint32_t get_judgment ()
- class Rel * get_next ()

5.13.1 Detailed Description

Rel A Relevant Result given by the TREC qrels file

Definition at line 7 of file Rel.h.

5.13.2 Constructor & Destructor Documentation

5.13.2.1 Rel() [1/2]

```
Rel::Rel ( )
```

Constructor 1 :default.

Definition at line 4 of file Rel.cpp.

5.13.2.2 Rel() [2/2]

```
Rel::Rel (
            char * c,
            uint32_t j )
```

Constructor 2.

Definition at line 7 of file Rel.cpp.

5.13.2.3 ~Rel()

```
Rel::~Rel ( )
```

Destructor.

Definition at line 20 of file Rel.cpp.

5.13.3 Member Function Documentation

5.13.3.1 display()

```
void Rel::display ( )
```

Definition at line 26 of file Rel.cpp.

5.13.3.2 get_code()

```
char * Rel::get_code ( )  [inline]
```

Accessors.

Definition at line 36 of file Rel.cpp.

5.13.3.3 get_judgment()

```
uint32_t Rel::get_judgment ( )  [inline]
```

Definition at line 37 of file Rel.cpp.

5.13.3.4 get_next()

```
class Rel * Rel::get_next ( )  [inline]
```

Definition at line 38 of file Rel.cpp.

5.13.3.5 set_code()

```
void Rel::set_code (
            char * v )  [inline]
```

Mutators.

Definition at line 32 of file Rel.cpp.

5.13.3.6 set_judgment()

```
void Rel::set_judgment (
            uint32_t v )  [inline]
```

Mutators.

Definition at line 31 of file Rel.cpp.

5.13.3.7 set_next()

```
void Rel::set_next (
            class Rel * v )  [inline]
```

Definition at line 33 of file Rel.cpp.

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

- FLAGR/src/Rel.h
- FLAGR/src/Rel.cpp

5.14 Rels Class Reference

`#include <Rels.h>`

Public Member Functions

- Rels ()
  Constructor 1 : Default.
- Rels (uint32_t)
  Constructor 2: overloaded.
- ~Rels ()
  Destructor.
- void display ()
- void insert (char *, uint32_t)
  Insert an element into the hash table.
- bool search (char *, uint32_t *)
  Search for an element in the hash table.
- uint32_t get_num_nodes ()
  Accessors.

5.14.1 Detailed Description

Rels A collection of Relevant Results (given by the TREC qrels file) for a specific query

Definition at line 7 of file Rels.h.

5.14.2 Constructor & Destructor Documentation

5.14.2.1 Rels() [1/2]

```
Rels::Rels ( )
```

Constructor 1 : Default.

Definition at line 4 of file Rels.cpp.

5.14.2.2 Rels() [2/2]

```
Rels::Rels (
            uint32_t size )
```

Constructor 2: overloaded.

Definition at line 7 of file Rels.cpp.

5.14.2.3 ~Rels()

```
Rels::~Rels ( )
```

Destructor.

Definition at line 20 of file Rels.cpp.

5.14.3 Member Function Documentation

5.14.3.1 display()

```
void Rels::display ( )
```

Definition at line 90 of file Rels.cpp.

5.14.3.2 get_num_nodes()

```
uint32_t Rels::get_num_nodes ( )  [inline]
```

Accessors.

Definition at line 88 of file Rels.cpp.

5.14.3.3 insert()

```
void Rels::insert (
            char * n,
            uint32_t j )
```

Insert an element into the hash table.

Find the hash value of the input term

Now search in the hash table to check whether this term exists or not

Traverse the linked list that represents the chain.

Return and exit

Create a new record and re-assign the linked list's head

Reassign the chain's head

Definition at line 36 of file Rels.cpp.

5.14.3.4 search()

```
bool Rels::search (
            char * n,
            uint32_t * r )
```

Search for an element in the hash table.

Find the hash value of the input term

Now search in the hash table to check whether this term exists or not.

Traverse the linked list that represents the chain.

Return and exit

Definition at line 66 of file Rels.cpp.

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

- FLAGR/src/Rels.h
- FLAGR/src/Rels.cpp

5.15 SimpleScoreStats Class Reference

```
#include <SimpleScoreStats.h>
```

Public Member Functions

- SimpleScoreStats ()
  Default (empty) constructor.
- ~SimpleScoreStats ()
  Destructor.
- void display ()
  Display the values.
- score_t get_min_val ()
  Accessors.
- score_t get_max_val ()
- score_t get_mean_val ()
- score_t get_std_val ()
- void set_min_val (score_t)
  Mutators.
- void set_max_val (score_t)
- void set_mean_val (score_t)
- void set_std_val (score_t)

5.15.1 Detailed Description

Definition at line 5 of file SimpleScoreStats.h.

5.15.2 Constructor & Destructor Documentation

5.15.2.1 SimpleScoreStats()

```
SimpleScoreStats::SimpleScoreStats ( )
```

Default (empty) constructor.

Definition at line 4 of file SimpleScoreStats.cpp.

5.15.2.2 ~SimpleScoreStats()

```
SimpleScoreStats::~SimpleScoreStats ( )
```

Destructor.

Definition at line 7 of file SimpleScoreStats.cpp.

5.15.3 Member Function Documentation

5.15.3.1 display()

```
void SimpleScoreStats::display ( )
```

Display the values.

Definition at line 12 of file SimpleScoreStats.cpp.

5.15.3.2 get_max_val()

```
score_t SimpleScoreStats::get_max_val ( )
```

Definition at line 19 of file SimpleScoreStats.cpp.

5.15.3.3 get_mean_val()

```
score_t SimpleScoreStats::get_mean_val ( )
```

Definition at line 20 of file SimpleScoreStats.cpp.

5.15.3.4 get_min_val()

```
score_t SimpleScoreStats::get_min_val ( )
```

Accessors.

Definition at line 18 of file SimpleScoreStats.cpp.

5.15.3.5 get_std_val()

```
score_t SimpleScoreStats::get_std_val ( )
```

Definition at line 21 of file SimpleScoreStats.cpp.

5.15.3.6 set_max_val()

```
void SimpleScoreStats::set_max_val (
            score_t v )
```

Definition at line 25 of file SimpleScoreStats.cpp.

5.15.3.7 set_mean_val()

```
void SimpleScoreStats::set_mean_val (
            score_t v )
```

Definition at line 26 of file SimpleScoreStats.cpp.

5.15.3.8 set_min_val()

```
void SimpleScoreStats::set_min_val (
            score_t v )
```

Mutators.

Definition at line 24 of file SimpleScoreStats.cpp.

5.15.3.9 set_std_val()

```
void SimpleScoreStats::set_std_val (
            score_t v )
```

Definition at line 27 of file SimpleScoreStats.cpp.

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

- FLAGR/src/SimpleScoreStats.h
- FLAGR/src/SimpleScoreStats.cpp

5.16 UserParams Struct Reference

External user parameters are passed to FLAGR via this UserParams structure.

`#include <InputParams.h>`

Public Attributes

- char * input_file = NULL
- char * rels_file = NULL
- char * random_string = NULL
- char * output_dir = NULL
- int eval_points = 0
- int rank_aggregation_method = 0
- int weight_normalization = 0
- int distance = 0
- float tol = 0.0
- int max_iter = 0
- bool prune = false
- bool exact = false
- score_t pref_thr = 0.0
- score_t veto_thr = 0.0
- score_t conc_thr = 0.0
- score_t disc_thr = 0.0
- score_t alpha = 0.0
- score_t beta = 0.0
- score_t gamma = 0.0
- score_t delta1 = 0.0
- score_t delta2 = 0.0
- score_t c1 = 0.0
- score_t c2 = 0.0

5.16.1 Detailed Description

External user parameters are passed to FLAGR via this UserParams structure.

Definition at line 5 of file InputParams.h.

5.16.2 Member Data Documentation

5.16.2.1 alpha

```
score_t UserParams::alpha = 0.0
```

Definition at line 27 of file InputParams.h.

5.16.2.2 beta

```
score_t UserParams::beta = 0.0
```

Definition at line 28 of file InputParams.h.

5.16.2.3 c1

```
score_t UserParams::c1 = 0.0
```

Definition at line 32 of file InputParams.h.

5.16.2.4 c2

```
score_t UserParams::c2 = 0.0
```

Definition at line 33 of file InputParams.h.

5.16.2.5 conc_thr

```
score_t UserParams::conc_thr = 0.0
```

Definition at line 24 of file InputParams.h.

5.16.2.6 delta1

```
score_t UserParams::delta1 = 0.0
```

Definition at line 30 of file InputParams.h.

5.16.2.7 delta2

```
score_t UserParams::delta2 = 0.0
```

Definition at line 31 of file InputParams.h.

5.16.2.8 disc_thr

```
score_t UserParams::disc_thr = 0.0
```

Definition at line 25 of file InputParams.h.

5.16.2.9 distance

```
int UserParams::distance = 0
```

Definition at line 15 of file InputParams.h.

5.16.2.10 eval_points

```
int UserParams::eval_points = 0
```

Definition at line 11 of file InputParams.h.

5.16.2.11 exact

```
bool UserParams::exact = false
```

Definition at line 20 of file InputParams.h.

5.16.2.12 gamma

```
score_t UserParams::gamma = 0.0
```

Definition at line 29 of file InputParams.h.

5.16.2.13 input_file

```
char* UserParams::input_file = NULL
```

Definition at line 6 of file InputParams.h.

5.16.2.14 max_iter

```
int UserParams::max_iter = 0
```

Definition at line 18 of file InputParams.h.

5.16.2.15 output_dir

```
char* UserParams::output_dir = NULL
```

Definition at line 9 of file InputParams.h.

5.16.2.16 pref_thr

```
score_t UserParams::pref_thr = 0.0
```

Definition at line 22 of file InputParams.h.

5.16.2.17 prune

```
bool UserParams::prune = false
```

Definition at line 19 of file InputParams.h.

5.16.2.18 random_string

```
char* UserParams::random_string = NULL
```

Definition at line 8 of file InputParams.h.

5.16.2.19 rank_aggregation_method

```
int UserParams::rank_aggregation_method = 0
```

Definition at line 13 of file InputParams.h.

5.16.2.20 rels_file

```
char* UserParams::rels_file = NULL
```

Definition at line 7 of file InputParams.h.

5.16.2.21 tol

```
float UserParams::tol = 0.0
```

Definition at line 17 of file InputParams.h.

5.16.2.22 veto_thr

```
score_t UserParams::veto_thr = 0.0
```

Definition at line 23 of file InputParams.h.

5.16.2.23 weight_normalization

```
int UserParams::weight_normalization = 0
```

Definition at line 14 of file InputParams.h.

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

- FLAGR/src/InputParams.h

5.17 Voter Class Reference

`#include <Voter.h>`

Public Member Functions

- Voter ()
  Constructor 1 : default.
- Voter (char *, score_t)
  Constructor 2.
- ~Voter ()
  Destructor.
- void display ()
  Display the Voter object.
- void set_name (char *)
  Mutators.
- void set_weight (score_t)
  Mutators.
- char * get_name ()
  Accessors.
- score_t get_weight ()

5.17.1 Detailed Description

Definition at line 5 of file Voter.h.

5.17.2 Constructor & Destructor Documentation

5.17.2.1 Voter() [1/2]

```
Voter::Voter ( )
```

Constructor 1 : default.

Definition at line 4 of file Voter.cpp.

5.17.2.2 Voter() [2/2]

```
Voter::Voter (
            char * n,
            score_t w )
```

Constructor 2.

Definition at line 7 of file Voter.cpp.

5.17.2.3 ~Voter()

```
Voter::~Voter ( )
```

Destructor.

Definition at line 12 of file Voter.cpp.

5.17.3 Member Function Documentation

5.17.3.1 display()

```
void Voter::display ( )
```

Display the Voter object.

Definition at line 19 of file Voter.cpp.

5.17.3.2 get_name()

```
char * Voter::get_name ( )
```

Accessors.

Definition at line 31 of file Voter.cpp.

5.17.3.3 get_weight()

```
score_t Voter::get_weight ( )
```

Definition at line 32 of file Voter.cpp.

5.17.3.4 set_name()

```
void Voter::set_name (
            char * v )
```

Mutators.

Definition at line 25 of file Voter.cpp.

5.17.3.5 set_weight()

```
void Voter::set_weight (
            score_t v )
```

Mutators.

Definition at line 24 of file Voter.cpp.

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

- FLAGR/src/Voter.h
- FLAGR/src/Voter.cpp

Chapter 6

File Documentation

6.1 FLAGR/cflagr.cpp File Reference

```
#include "driver.cpp"
```

Functions

- void Linear (const char inf[ ], const char relf[ ], const int evpts, const int ram, const char ranstr[ ], const char out[ ])

  Wrapper for CombSUM/CombMNZ.
- void Condorcet (const char inf[ ], const char relf[ ], const int evpts, const char ranstr[ ], const char out[ ])

  Wrapper for the Condorcet Winners Method.
- void Copeland (const char inf[ ], const char relf[ ], const int evpts, const char ranstr[ ], const char out[ ])

  Wrapper for the Copeland Winners Method.
- void OutrankingApproach (const char inf[ ], const char relf[ ], const int evpts, const char ranstr[ ], const char out[ ], const float pref_t, const float veto_t, const float conc_t, const float disc_t)

  Wrapper for the Outranking Approach of Farah and Vanderpooten [2].
- void Kemeny (const char inf[ ], const char relf[ ], const int evpts, const char ranstr[ ], const char out[ ])

  Wrapper for Kemeny Optimal Aggregation (brute force implementation)
- void RobustRA (const char inf[ ], const char relf[ ], const int evpts, const char ranstr[ ], const char out[ ], const bool exact)

  Wrapper for Robust Rank Aggregation of Kolde et. al, 2012 [7].
- void DIBRA (const char inf[ ], const char relf[ ], const int evpts, const int agg, const char ranstr[ ], const char out[ ], const int wnorm, const int dist, const bool prune, const float gamma, const float d1, const float d2, const float tol, const int iter, const float pref_t, const float veto_t, const float conc_t, const float disc_t)

  Wrapper the distance-based iterative rank aggregation method of Akritidis et. al [5] - DIBRA.
- void PrefRel (const char inf[ ], const char relf[ ], const int evpts, const char ranstr[ ], const char out[ ], const float alpha, const float beta)

  Wrapper the preference relation rank aggregation method of Desarkar et. al, 2016 [3].
- void Agglomerative (const char inf[ ], const char relf[ ], const int evpts, const char ranstr[ ], const char out[ ], const float c1, const float c2)

  Wrapper for Agglomerative rank aggregation method of Chatterjee et. al, 2018 [4].
- void MC (const char inf[ ], const char relf[ ], const int evpts, const int ram, const char ranstr[ ], const char out[ ], const float ergodic_number, const float delta, const int iter)

  Wrapper for the Markov Chains methods of Dwork et. al, 2001 [6].
- void Custom1 (const char inf[ ], const char relf[ ], const int evpts, const char ranstr[ ], const char out[ ])

  Wrapper for the first custom method.
- void Custom2 (const char inf[ ], const char relf[ ], const int evpts, const char ranstr[ ], const char out[ ])

  Wrapper for the second custom method.

6.1.1 Function Documentation

6.1.1.1 Agglomerative()

```
void Agglomerative (
            const char inf[],
            const char relf[],
            const int evpts,
            const char ranstr[],
            const char out[],
            const float c1,
            const float c2 )
```

Wrapper for Agglomerative rank aggregation method of Chatterjee et. al, 2018 [4].

Definition at line 300 of file cflagr.cpp.

6.1.1.2 Condorcet()

```
void Condorcet (
            const char inf[],
            const char relf[],
            const int evpts,
            const char ranstr[],
            const char out[] )
```

Wrapper for the Condorcet Winners Method.

Definition at line 41 of file cflagr.cpp.

6.1.1.3 Copeland()

```
void Copeland (
            const char inf[],
            const char relf[],
            const int evpts,
            const char ranstr[],
            const char out[] )
```

Wrapper for the Copeland Winners Method.

Definition at line 74 of file cflagr.cpp.

6.1.1.4 Custom1()

```
void Custom1 (
             const char inf[],
             const char relf[],
             const int evpts,
             const char ranstr[],
             const char out[] )
```

Wrapper for the first custom method.

Definition at line 373 of file cflagr.cpp.

6.1.1.5 Custom2()

```
void Custom2 (
             const char inf[],
             const char relf[],
             const int evpts,
             const char ranstr[],
             const char out[] )
```

Wrapper for the second custom method.

Definition at line 406 of file cflagr.cpp.

6.1.1.6 DIBRA()

```
void DIBRA (
             const char inf[],
             const char relf[],
             const int evpts,
             const int agg,
             const char ranstr[],
             const char out[],
             const int wnorm,
             const int dist,
             const bool prune,
             const float gamma,
             const float d1,
             const float d2,
             const float tol,
             const int iter,
             const float pref_t,
             const float veto_t,
             const float conc_t,
             const float disc_t )
```

Wrapper the distance-based iterative rank aggregation method of Akritidis et. al [5] - DIBRA.

Definition at line 215 of file cflagr.cpp.

6.1.1.7 Kemeny()

```
void Kemeny (
             const char inf[],
             const char relf[],
             const int evpts,
             const char ranstr[],
             const char out[] )
```

Wrapper for Kemeny Optimal Aggregation (brute force implementation)

Definition at line 146 of file cflagr.cpp.

6.1.1.8 Linear()

```
void Linear (
             const char inf[],
             const char relf[],
             const int evpts,
             const int ram,
             const char ranstr[],
             const char out[] )
```

Wrapper for CombSUM/CombMNZ.

// FLAGR exposed C functions - Dynamic Library References C functions that i) wrap around the corresponding C++ functions and ii) are exposed to shared library.

Definition at line 8 of file cflagr.cpp.

6.1.1.9 MC()

```
void MC (
             const char inf[],
             const char relf[],
             const int evpts,
             const int ram,
             const char ranstr[],
             const char out[],
             const float ergodic_number,
             const float delta,
             const int iter )
```

Wrapper for the Markov Chains methods of Dwork et. al, 2001 [6].

Definition at line 336 of file cflagr.cpp.

6.1.1.10 OutrankingApproach()

```
void OutrankingApproach (
             const char inf[],
             const char relf[],
             const int evpts,
             const char ranstr[],
             const char out[],
             const float pref_t,
             const float veto_t,
             const float conc_t,
             const float disc_t )
```

Wrapper for the Outranking Approach of Farah and Vanderpooten [2].

Definition at line 107 of file cflagr.cpp.

6.1.1.11 PrefRel()

```
void PrefRel (
             const char inf[],
             const char relf[],
             const int evpts,
             const char ranstr[],
             const char out[],
             const float alpha,
             const float beta )
```

Wrapper the preference relation rank aggregation method of Desarkar et. al, 2016 [3].

Definition at line 264 of file cflagr.cpp.

6.1.1.12 RobustRA()

```
void RobustRA (
             const char inf[],
             const char relf[],
             const int evpts,
             const char ranstr[],
             const char out[],
             const bool exact )
```

Wrapper for Robust Rank Aggregation of Kolde et. al, 2012 [7].

Definition at line 179 of file cflagr.cpp.

6.2 cflagr.cpp

Go to the documentation of this file.

```
00001 #include "driver.cpp"
00002
00006 extern "C" {
00008     void Linear(const char inf[], const char relf[], const int evpts, const int ram, const char
      ranstr[], const char out[]) {
00009
00010         struct UserParams uParams;
00011         srand(time(0));
00012
00013         uParams.input_file = (char *)malloc((strlen(inf) + 1) * sizeof(char));
00014         strcpy(uParams.input_file, inf);
00015
00016         uParams.random_string = (char *)malloc((strlen(ranstr) + 1) * sizeof(char));
00017         strcpy(uParams.random_string, ranstr);
00018
00019         uParams.output_dir = (char *)malloc((strlen(out) + 1) * sizeof(char));
00020         strcpy(uParams.output_dir, out);
00021
00022         if (strlen(relf) > 0) {
00023             uParams.rels_file = (char *)malloc((strlen(relf) + 1) * sizeof(char));
00024             strcpy(uParams.rels_file, relf);
00025         } else {
00026             uParams.rels_file = NULL;
00027         }
00028
00029         uParams.eval_points = evpts;
00030         uParams.rank_aggregation_method = ram;
00031
00032         FLAGR_DRIVER(uParams);
00033
00034         if (uParams.input_file) free(uParams.input_file);
00035         if (uParams.rels_file) free(uParams.rels_file);
00036         if (uParams.rels_file) free(uParams.random_string);
00037         if (uParams.output_dir) free(uParams.output_dir);
00038     }
00039
00041     void Condorcet(const char inf[], const char relf[], const int evpts, const char ranstr[], const
      char out[]) {
00042
00043         struct UserParams uParams;
00044         srand(time(0));
00045
00046         uParams.input_file = (char *)malloc((strlen(inf) + 1) * sizeof(char));
00047         strcpy(uParams.input_file, inf);
00048
00049         uParams.random_string = (char *)malloc((strlen(ranstr) + 1) * sizeof(char));
00050         strcpy(uParams.random_string, ranstr);
00051
00052         uParams.output_dir = (char *)malloc((strlen(out) + 1) * sizeof(char));
00053         strcpy(uParams.output_dir, out);
00054
00055         if (strlen(relf) > 0) {
00056             uParams.rels_file = (char *)malloc((strlen(relf) + 1) * sizeof(char));
00057             strcpy(uParams.rels_file, relf);
00058         } else {
00059             uParams.rels_file = NULL;
00060         }
00061
00062         uParams.eval_points = evpts;
00063         uParams.rank_aggregation_method = 200;
00064
00065         FLAGR_DRIVER(uParams);
00066
00067         if (uParams.input_file) free(uParams.input_file);
00068         if (uParams.rels_file) free(uParams.rels_file);
00069         if (uParams.rels_file) free(uParams.random_string);
00070         if (uParams.output_dir) free(uParams.output_dir);
00071     }
00072
00074     void Copeland(const char inf[], const char relf[], const int evpts, const char ranstr[], const
      char out[]) {
00075
00076         struct UserParams uParams;
00077         srand(time(0));
00078
00079         uParams.input_file = (char *)malloc((strlen(inf) + 1) * sizeof(char));
00080         strcpy(uParams.input_file, inf);
00081
00082         uParams.random_string = (char *)malloc((strlen(ranstr) + 1) * sizeof(char));
00083         strcpy(uParams.random_string, ranstr);
00084
00085         uParams.output_dir = (char *)malloc((strlen(out) + 1) * sizeof(char));
```

```
00086         strcpy(uParams.output_dir, out);
00087 
00088         if (strlen(relf) > 0) {
00089             uParams.rels_file = (char *)malloc((strlen(relf) + 1) * sizeof(char));
00090             strcpy(uParams.rels_file, relf);
00091         } else {
00092             uParams.rels_file = NULL;
00093         }
00094 
00095         uParams.eval_points = evpts;
00096         uParams.rank_aggregation_method = 201;
00097 
00098         FLAGR_DRIVER(uParams);
00099 
00100         if (uParams.input_file) free(uParams.input_file);
00101         if (uParams.rels_file) free(uParams.rels_file);
00102         if (uParams.rels_file) free(uParams.random_string);
00103         if (uParams.output_dir) free(uParams.output_dir);
00104     }
00105 
00107     void OutrankingApproach
00108         (const char inf[], const char relf[], const int evpts, const char ranstr[], const char out[],
00109         const float pref_t, const float veto_t, const float conc_t, const float disc_t) {
00110 
00111             struct UserParams uParams;
00112             srand(time(0));
00113 
00114             uParams.input_file = (char *)malloc((strlen(inf) + 1) * sizeof(char));
00115             strcpy(uParams.input_file, inf);
00116 
00117             uParams.random_string = (char *)malloc((strlen(ranstr) + 1) * sizeof(char));
00118             strcpy(uParams.random_string, ranstr);
00119 
00120             uParams.output_dir = (char *)malloc((strlen(out) + 1) * sizeof(char));
00121             strcpy(uParams.output_dir, out);
00122 
00123             if (strlen(relf) > 0) {
00124                 uParams.rels_file = (char *)malloc((strlen(relf) + 1) * sizeof(char));
00125                 strcpy(uParams.rels_file, relf);
00126             } else {
00127                 uParams.rels_file = NULL;
00128             }
00129 
00130             uParams.eval_points = evpts;
00131             uParams.rank_aggregation_method = 300;
00132             uParams.pref_thr = pref_t;
00133             uParams.veto_thr = veto_t;
00134             uParams.conc_thr = conc_t;
00135             uParams.disc_thr = disc_t;
00136 
00137             FLAGR_DRIVER(uParams);
00138 
00139             if (uParams.input_file) free(uParams.input_file);
00140             if (uParams.rels_file) free(uParams.rels_file);
00141             if (uParams.rels_file) free(uParams.random_string);
00142             if (uParams.output_dir) free(uParams.output_dir);
00143     }
00144 
00146     void Kemeny(const char inf[], const char relf[], const int evpts, const char ranstr[], const char
      out[]) {
00147 
00148         struct UserParams uParams;
00149         srand(time(0));
00150 
00151         uParams.input_file = (char *)malloc((strlen(inf) + 1) * sizeof(char));
00152         strcpy(uParams.input_file, inf);
00153 
00154         uParams.random_string = (char *)malloc((strlen(ranstr) + 1) * sizeof(char));
00155         strcpy(uParams.random_string, ranstr);
00156 
00157         uParams.output_dir = (char *)malloc((strlen(out) + 1) * sizeof(char));
00158         strcpy(uParams.output_dir, out);
00159 
00160         if (strlen(relf) > 0) {
00161             uParams.rels_file = (char *)malloc((strlen(relf) + 1) * sizeof(char));
00162             strcpy(uParams.rels_file, relf);
00163         } else {
00164             uParams.rels_file = NULL;
00165         }
00166 
00167         uParams.eval_points = evpts;
00168         uParams.rank_aggregation_method = 400;
00169 
00170         FLAGR_DRIVER(uParams);
00171 
00172         if (uParams.input_file) free(uParams.input_file);
00173         if (uParams.rels_file) free(uParams.rels_file);
```

```
00174         if (uParams.rels_file) free(uParams.random_string);
00175         if (uParams.output_dir) free(uParams.output_dir);
00176     }
00177
00179     void RobustRA
00180         (const char inf[], const char relf[], const int evpts, const char ranstr[], const char out[],
00181         const bool exact) {
00182
00183             struct UserParams uParams;
00184             srand(time(0));
00185
00186             uParams.input_file = (char *)malloc((strlen(inf) + 1) * sizeof(char));
00187             strcpy(uParams.input_file, inf);
00188
00189             uParams.random_string = (char *)malloc((strlen(ranstr) + 1) * sizeof(char));
00190             strcpy(uParams.random_string, ranstr);
00191
00192             uParams.output_dir = (char *)malloc((strlen(out) + 1) * sizeof(char));
00193             strcpy(uParams.output_dir, out);
00194
00195             if (strlen(relf) > 0) {
00196                 uParams.rels_file = (char *)malloc((strlen(relf) + 1) * sizeof(char));
00197                 strcpy(uParams.rels_file, relf);
00198             } else {
00199                 uParams.rels_file = NULL;
00200             }
00201
00202             uParams.eval_points = evpts;
00203             uParams.exact = exact;
00204             uParams.rank_aggregation_method = 401;
00205
00206             FLAGR_DRIVER(uParams);
00207
00208             if (uParams.input_file) free(uParams.input_file);
00209             if (uParams.rels_file) free(uParams.rels_file);
00210             if (uParams.rels_file) free(uParams.random_string);
00211             if (uParams.output_dir) free(uParams.output_dir);
00212     }
00213
00215     void DIBRA
00216         (const char inf[], const char relf[], const int evpts, const int agg, const char ranstr[],
00217         const char out[], const int wnorm, const int dist, const bool prune, const float gamma,
00218         const float d1, const float d2, const float tol, const int iter,
00219         const float pref_t, const float veto_t, const float conc_t, const float disc_t) {
00220
00221             struct UserParams uParams;
00222             srand(time(0));
00223
00224             uParams.input_file = (char *)malloc((strlen(inf) + 1) * sizeof(char));
00225             strcpy(uParams.input_file, inf);
00226
00227             uParams.random_string = (char *)malloc((strlen(ranstr) + 1) * sizeof(char));
00228             strcpy(uParams.random_string, ranstr);
00229
00230             uParams.output_dir = (char *)malloc((strlen(out) + 1) * sizeof(char));
00231             strcpy(uParams.output_dir, out);
00232
00233             if (strlen(relf) > 0) {
00234                 uParams.rels_file = (char *)malloc((strlen(relf) + 1) * sizeof(char));
00235                 strcpy(uParams.rels_file, relf);
00236             } else {
00237                 uParams.rels_file = NULL;
00238             }
00239
00240             uParams.eval_points = evpts;
00241             uParams.rank_aggregation_method = agg;
00242             uParams.weight_normalization = wnorm;
00243             uParams.distance = dist;
00244             uParams.prune = prune;
00245             uParams.gamma = gamma;
00246             uParams.delta1 = d1;
00247             uParams.delta2 = d2;
00248             uParams.tol = tol;
00249             uParams.max_iter = iter;
00250             uParams.pref_thr = pref_t;
00251             uParams.veto_thr = veto_t;
00252             uParams.conc_thr = conc_t;
00253             uParams.disc_thr = disc_t;
00254
00255             FLAGR_DRIVER(uParams);
00256
00257             if (uParams.input_file) free(uParams.input_file);
00258             if (uParams.rels_file) free(uParams.rels_file);
00259             if (uParams.rels_file) free(uParams.random_string);
00260             if (uParams.output_dir) free(uParams.output_dir);
00261     }
00262
```

```
00264     void PrefRel(const char inf[], const char relf[], const int evpts, const char ranstr[],
00265         const char out[], const float alpha, const float beta) {
00266 
00267             struct UserParams uParams;
00268             srand(time(0));
00269 
00270             uParams.input_file = (char *)malloc((strlen(inf) + 1) * sizeof(char));
00271             strcpy(uParams.input_file, inf);
00272 
00273             uParams.random_string = (char *)malloc((strlen(ranstr) + 1) * sizeof(char));
00274             strcpy(uParams.random_string, ranstr);
00275 
00276             uParams.output_dir = (char *)malloc((strlen(out) + 1) * sizeof(char));
00277             strcpy(uParams.output_dir, out);
00278 
00279             if (strlen(relf) > 0) {
00280                 uParams.rels_file = (char *)malloc((strlen(relf) + 1) * sizeof(char));
00281                 strcpy(uParams.rels_file, relf);
00282             } else {
00283                 uParams.rels_file = NULL;
00284             }
00285 
00286             uParams.eval_points = evpts;
00287             uParams.rank_aggregation_method = 600;
00288             uParams.alpha = alpha;
00289             uParams.beta = beta;
00290 
00291             FLAGR_DRIVER(uParams);
00292 
00293             if (uParams.input_file) free(uParams.input_file);
00294             if (uParams.rels_file) free(uParams.rels_file);
00295             if (uParams.rels_file) free(uParams.random_string);
00296             if (uParams.output_dir) free(uParams.output_dir);
00297     }
00298 
00300     void Agglomerative(const char inf[], const char relf[], const int evpts, const char ranstr[],
00301         const char out[], const float c1, const float c2) {
00302 
00303             struct UserParams uParams;
00304             srand(time(0));
00305 
00306             uParams.input_file = (char *)malloc((strlen(inf) + 1) * sizeof(char));
00307             strcpy(uParams.input_file, inf);
00308 
00309             uParams.random_string = (char *)malloc((strlen(ranstr) + 1) * sizeof(char));
00310             strcpy(uParams.random_string, ranstr);
00311 
00312             uParams.output_dir = (char *)malloc((strlen(out) + 1) * sizeof(char));
00313             strcpy(uParams.output_dir, out);
00314 
00315             if (strlen(relf) > 0) {
00316                 uParams.rels_file = (char *)malloc((strlen(relf) + 1) * sizeof(char));
00317                 strcpy(uParams.rels_file, relf);
00318             } else {
00319                 uParams.rels_file = NULL;
00320             }
00321 
00322             uParams.eval_points = evpts;
00323             uParams.rank_aggregation_method = 700;
00324             uParams.c1 = c1;
00325             uParams.c2 = c2;
00326 
00327             FLAGR_DRIVER(uParams);
00328 
00329             if (uParams.input_file) free(uParams.input_file);
00330             if (uParams.rels_file) free(uParams.rels_file);
00331             if (uParams.rels_file) free(uParams.random_string);
00332             if (uParams.output_dir) free(uParams.output_dir);
00333     }
00334 
00336     void MC(const char inf[], const char relf[], const int evpts, const int ram, const char ranstr[],
00337         const char out[], const float ergodic_number, const float delta, const int iter) {
00338 
00339             struct UserParams uParams;
00340             srand(time(0));
00341 
00342             uParams.input_file = (char *)malloc((strlen(inf) + 1) * sizeof(char));
00343             strcpy(uParams.input_file, inf);
00344 
00345             uParams.random_string = (char *)malloc((strlen(ranstr) + 1) * sizeof(char));
00346             strcpy(uParams.random_string, ranstr);
00347 
00348             uParams.output_dir = (char *)malloc((strlen(out) + 1) * sizeof(char));
00349             strcpy(uParams.output_dir, out);
00350 
00351             if (strlen(relf) > 0) {
00352                 uParams.rels_file = (char *)malloc((strlen(relf) + 1) * sizeof(char));
```

```
00353                 strcpy(uParams.rels_file, relf);
00354             } else {
00355                 uParams.rels_file = NULL;
00356             }
00357 
00358             uParams.eval_points = evpts;
00359             uParams.rank_aggregation_method = ram;
00360             uParams.alpha = ergodic_number;
00361             uParams.delta1 = delta;
00362             uParams.max_iter = iter;
00363 
00364             FLAGR_DRIVER(uParams);
00365 
00366             if (uParams.input_file) free(uParams.input_file);
00367             if (uParams.rels_file) free(uParams.rels_file);
00368             if (uParams.rels_file) free(uParams.random_string);
00369             if (uParams.output_dir) free(uParams.output_dir);
00370     }
00371 
00373     void Custom1(const char inf[], const char relf[], const int evpts, const char ranstr[], const char
    out[]) {
00374 
00375         struct UserParams uParams;
00376         srand(time(0));
00377 
00378         uParams.input_file = (char *)malloc((strlen(inf) + 1) * sizeof(char));
00379         strcpy(uParams.input_file, inf);
00380 
00381         uParams.random_string = (char *)malloc((strlen(ranstr) + 1) * sizeof(char));
00382         strcpy(uParams.random_string, ranstr);
00383 
00384         uParams.output_dir = (char *)malloc((strlen(out) + 1) * sizeof(char));
00385         strcpy(uParams.output_dir, out);
00386 
00387         if (strlen(relf) > 0) {
00388             uParams.rels_file = (char *)malloc((strlen(relf) + 1) * sizeof(char));
00389             strcpy(uParams.rels_file, relf);
00390         } else {
00391             uParams.rels_file = NULL;
00392         }
00393 
00394         uParams.eval_points = evpts;
00395         uParams.rank_aggregation_method = 901;
00396 
00397         FLAGR_DRIVER(uParams);
00398 
00399         if (uParams.input_file) free(uParams.input_file);
00400         if (uParams.rels_file) free(uParams.rels_file);
00401         if (uParams.rels_file) free(uParams.random_string);
00402         if (uParams.output_dir) free(uParams.output_dir);
00403     }
00404 
00406     void Custom2(const char inf[], const char relf[], const int evpts, const char ranstr[], const char
    out[]) {
00407 
00408         struct UserParams uParams;
00409         srand(time(0));
00410 
00411         uParams.input_file = (char *)malloc((strlen(inf) + 1) * sizeof(char));
00412         strcpy(uParams.input_file, inf);
00413 
00414         uParams.random_string = (char *)malloc((strlen(ranstr) + 1) * sizeof(char));
00415         strcpy(uParams.random_string, ranstr);
00416 
00417         uParams.output_dir = (char *)malloc((strlen(out) + 1) * sizeof(char));
00418         strcpy(uParams.output_dir, out);
00419 
00420         if (strlen(relf) > 0) {
00421             uParams.rels_file = (char *)malloc((strlen(relf) + 1) * sizeof(char));
00422             strcpy(uParams.rels_file, relf);
00423         } else {
00424             uParams.rels_file = NULL;
00425         }
00426 
00427         uParams.eval_points = evpts;
00428         uParams.rank_aggregation_method = 902;
00429 
00430         FLAGR_DRIVER(uParams);
00431 
00432         if (uParams.input_file) free(uParams.input_file);
00433         if (uParams.rels_file) free(uParams.rels_file);
00434         if (uParams.rels_file) free(uParams.random_string);
00435         if (uParams.output_dir) free(uParams.output_dir);
00436     }
00437 }
00438 
```

6.3 FLAGR/dllflagr.cpp File Reference

```
#include "driver.cpp"
```

Functions

- __declspec (dllexport) void __cdecl Linear(const char inf[ ]
  Wrapper for CombSUM/CombMNZ.
- srand (time(0))
- strcpy (uParams.input_file, inf)
- strcpy (uParams.random_string, ranstr)
- strcpy (uParams.output_dir, out)
- if (strlen(relf) > 0)
- FLAGR_DRIVER (uParams)
- if (uParams.input_file) free(uParams.input_file)

Variables

- const char relf [ ]
- const char const int evpts
- const char const int const int ram
- const char const int const int const char ranstr [ ]
- const char const int const int const char const char out [ ]
- uParams input_file = (char *)malloc((strlen(inf) + 1) * sizeof(char))
- uParams random_string = (char *)malloc((strlen(ranstr) + 1) * sizeof(char))
- uParams output_dir = (char *)malloc((strlen(out) + 1) * sizeof(char))
- else
- uParams eval_points = evpts
- uParams rank_aggregation_method = ram
- const char const int const char const char const float pref_t
- const char const int const char const char const float const float veto_t
- const char const int const char const char const float const float const float conc_t
- const char const int const char const char const float const float const float const float disc_t
- uParams pref_thr = pref_t
- uParams veto_thr = veto_t
- uParams conc_thr = conc_t
- uParams disc_thr = disc_t
- const char const int const char const char const bool exact
- const char const int const int agg
- const char const int const int const char const char const int wnorm
- const char const int const int const char const char const int const int dist
- const char const int const int const char const char const int const int const bool prune = prune
- const char const int const int const char const char const int const int const bool const float gamma = gamma
- const char const int const int const char const char const int const int const bool const float const float d1
- const char const int const int const char const char const int const int const bool const float const float const float d2
- const char const int const int const char const char const int const int const bool const float const float const float const float tol = tol
- const char const int const int const char const char const int const int const bool const float const float const float const float const int iter
- uParams weight_normalization = wnorm

- uParams distance = dist
- uParams delta1 = d1
- uParams delta2 = d2
- uParams max_iter = iter
- const char const int const char const char const float alpha = alpha
- const char const int const char const char const float const float beta
- const char const int const char const char const float c1 = c1
- const char const int const char const char const float const float c2
- const char const int const int const char const char const float ergodic_number
- const char const int const int const char const char const float const float delta

6.3.1 Function Documentation

6.3.1.1 __declspec()

```
__declspec (
            dllexport  ) const
```

Wrapper for CombSUM/CombMNZ.

Wrapper for the second custom method.

Wrapper for the first custom method.

Wrapper for the Markov Chains methods of Dwork et. al, 2001 [6].

Wrapper for Agglomerative rank aggregation method of Chatterjee et. al, 2018 [4].

Wrapper the preference relation rank aggregation method of Desarkar et. al, 2016 [3].

Wrapper the distance-based iterative rank aggregation method of Akritidis et. al [5] - DIBRA.

Wrapper for Robust Rank Aggregation of Kolde et. al, 2012 [7].

Wrapper for Kemeny Optimal Aggregation (brute force implementation)

Wrapper for the Outranking Approach of Farah and Vanderpooten [2].

Wrapper for the Copeland Winners Method.

Wrapper for the Condorcet Winners Method.

// FLAGR exposed C functions - Dynamic Library References C functions that i) wrap around the corresponding C++ functions and ii) are exposed to DLL.

6.3.1.2 FLAGR_DRIVER()

```
FLAGR_DRIVER (
            uParams  )
```

6.3.1.3 if() [1/2]

```
if (
            strlen(relf) ,
            0  )
```

Definition at line 22 of file dllflagr.cpp.

6.3.1.4 if() [2/2]

```
if (
            uParams.  input_file )
```

6.3.1.5 srand()

```
srand (
            time(0)  )
```

6.3.1.6 strcpy() [1/3]

```
strcpy (
            uParams.  input_file,
            inf  )
```

6.3.1.7 strcpy() [2/3]

```
strcpy (
            uParams.  output_dir,
            out  )
```

6.3.1.8 strcpy() [3/3]

```
strcpy (
            uParams.  random_string,
            ranstr  )
```

6.3.2 Variable Documentation

6.3.2.1 agg

```
const char const int const int agg
```

Definition at line 216 of file dllflagr.cpp.

6.3.2.2 alpha

```
uParams alpha = alpha
```

Definition at line 265 of file dllflagr.cpp.

6.3.2.3 beta

```
uParams beta
```

Initial value:

```
{
                struct UserParams uParams
```

Definition at line 265 of file dllflagr.cpp.

6.3.2.4 c1

```
uParams c1 = c1
```

Definition at line 301 of file dllflagr.cpp.

6.3.2.5 c2

```
uParams c2
```

Initial value:

```
{
                struct UserParams uParams
```

Definition at line 301 of file dllflagr.cpp.

6.3.2.6 conc_t

```
const char const int const int const char const char const int const int const bool const
float const float const float const float const int const float const float const float conc↩
_t
```

Definition at line 109 of file dllflagr.cpp.

6.3.2.7 conc_thr

```
uParams conc_thr = conc_t
```

Definition at line 134 of file dllflagr.cpp.

6.3.2.8 d1

```
const char const int const int const char const char const int const int const bool const
float const float d1
```

Definition at line 218 of file dllflagr.cpp.

6.3.2.9 d2

```
const char const int const int const char const char const int const int const bool const
float const float const float d2
```

Definition at line 218 of file dllflagr.cpp.

6.3.2.10 delta

```
const char const int const int const char const char const float const float delta
```

Definition at line 337 of file dllflagr.cpp.

6.3.2.11 delta1

```
uParams delta1 = d1
```

Definition at line 246 of file dllflagr.cpp.

6.3.2.12 delta2

```
uParams delta2 = d2
```

Definition at line 247 of file dllflagr.cpp.

6.3.2.13 disc_t

```
const char const int const int const char const char const int const int const bool const
float const float const float const float const int const float const float const float const
float disc_t
```

Initial value:

```
{
                struct UserParams uParams
```

Definition at line 109 of file dllflagr.cpp.

6.3.2.14 disc_thr

```
uParams disc_thr = disc_t
```

Definition at line 135 of file dllflagr.cpp.

6.3.2.15 dist

```
const char const int const int const char const char const int const int dist
```

Definition at line 217 of file dllflagr.cpp.

6.3.2.16 distance

```
uParams distance = dist
```

Definition at line 243 of file dllflagr.cpp.

6.3.2.17 else

```
else
```

Initial value:

```
{
            uParams.rels_file = NULL
```

Definition at line 25 of file dllflagr.cpp.

6.3.2.18 ergodic_number

```
const char const int const int const char const char const float ergodic_number
```

Definition at line 337 of file dllflagr.cpp.

6.3.2.19 eval_points

```
uParams eval_points = evpts
```

Definition at line 29 of file dllflagr.cpp.

6.3.2.20 evpts

```
const char const int evpts
```

Definition at line 8 of file dllflagr.cpp.

6.3.2.21 exact

```
uParams exact
```

Initial value:

```
{
            struct UserParams uParams
```

Definition at line 181 of file dllflagr.cpp.

6.3.2.22 gamma

```
uParams gamma = gamma
```

Definition at line 217 of file dllflagr.cpp.

6.3.2.23 input_file

```
uParams input_file = (char *)malloc((strlen(inf) + 1) * sizeof(char))
```

Definition at line 13 of file dllflagr.cpp.

6.3.2.24 iter

```
const char const int const int const char const char const float const float const int iter
```

Initial value:

```
{
                struct UserParams uParams
```

Definition at line 218 of file dllflagr.cpp.

6.3.2.25 max_iter

```
uParams max_iter = iter
```

Definition at line 249 of file dllflagr.cpp.

6.3.2.26 out

```
const char const int const char const char out
```

Initial value:

```
{
        struct UserParams uParams
```

Definition at line 8 of file dllflagr.cpp.

6.3.2.27 output_dir

```
uParams output_dir = (char *)malloc((strlen(out) + 1) * sizeof(char))
```

Definition at line 19 of file dllflagr.cpp.

6.3.2.28 pref_t

```
const char const int const int const char const char const int const int const bool const
float const float const float const float const int const float pref_t
```

Definition at line 109 of file dllflagr.cpp.

6.3.2.29 pref_thr

```
uParams pref_thr = pref_t
```

Definition at line 132 of file dllflagr.cpp.

6.3.2.30 prune

```
uParams prune = prune
```

Definition at line 217 of file dllflagr.cpp.

6.3.2.31 ram

```
const char const int const int ram
```

Definition at line 8 of file dllflagr.cpp.

6.3.2.32 random_string

```
uParams random_string = (char *)malloc((strlen(ranstr) + 1) * sizeof(char))
```

Definition at line 16 of file dllflagr.cpp.

6.3.2.33 rank_aggregation_method

```
uParams rank_aggregation_method = ram
```

Definition at line 30 of file dllflagr.cpp.

6.3.2.34 ranstr

```
const char const int const char ranstr
```

Definition at line 8 of file dllflagr.cpp.

6.3.2.35 relf

```
const char relf
```

Definition at line 8 of file dllflagr.cpp.

6.3.2.36 tol

```
uParams tol = tol
```

Definition at line 218 of file dllflagr.cpp.

6.3.2.37 veto_t

```
const char const int const int const char const char const int const int const bool const
float const float const float const float const int const float const float veto_t
```

Definition at line 109 of file dllflagr.cpp.

6.3.2.38 veto_thr

```
uParams veto_thr = veto_t
```

Definition at line 133 of file dllflagr.cpp.

6.3.2.39 weight_normalization

```
uParams weight_normalization = wnorm
```

Definition at line 242 of file dllflagr.cpp.

6.3.2.40 wnorm

```
const char const int const int const char const char const int wnorm
```

Definition at line 217 of file dllflagr.cpp.

6.4 dllflagr.cpp

Go to the documentation of this file.

```
00001 #include "driver.cpp"
00002
00006 extern "C" {
00008     __declspec(dllexport) void __cdecl Linear(const char inf[], const char relf[], const int evpts,
      const int ram, const char ranstr[], const char out[]) {
00009
00010         struct UserParams uParams;
00011         srand(time(0));
00012
00013         uParams.input_file = (char *)malloc((strlen(inf) + 1) * sizeof(char));
00014         strcpy(uParams.input_file, inf);
00015
00016         uParams.random_string = (char *)malloc((strlen(ranstr) + 1) * sizeof(char));
00017         strcpy(uParams.random_string, ranstr);
00018
00019         uParams.output_dir = (char *)malloc((strlen(out) + 1) * sizeof(char));
00020         strcpy(uParams.output_dir, out);
00021
00022         if (strlen(relf) > 0) {
00023             uParams.rels_file = (char *)malloc((strlen(relf) + 1) * sizeof(char));
00024             strcpy(uParams.rels_file, relf);
00025         } else {
00026             uParams.rels_file = NULL;
00027         }
00028
00029         uParams.eval_points = evpts;
00030         uParams.rank_aggregation_method = ram;
00031
00032         FLAGR_DRIVER(uParams);
00033
00034         if (uParams.input_file) free(uParams.input_file);
00035         if (uParams.rels_file) free(uParams.rels_file);
00036         if (uParams.rels_file) free(uParams.random_string);
00037         if (uParams.output_dir) free(uParams.output_dir);
00038     }
00039
00041     __declspec(dllexport) void __cdecl Condorcet(const char inf[], const char relf[], const int evpts,
      const char ranstr[], const char out[]) {
00042
00043         struct UserParams uParams;
00044         srand(time(0));
00045
00046         uParams.input_file = (char *)malloc((strlen(inf) + 1) * sizeof(char));
00047         strcpy(uParams.input_file, inf);
00048
00049         uParams.random_string = (char *)malloc((strlen(ranstr) + 1) * sizeof(char));
00050         strcpy(uParams.random_string, ranstr);
00051
00052         uParams.output_dir = (char *)malloc((strlen(out) + 1) * sizeof(char));
00053         strcpy(uParams.output_dir, out);
00054
00055         if (strlen(relf) > 0) {
00056             uParams.rels_file = (char *)malloc((strlen(relf) + 1) * sizeof(char));
00057             strcpy(uParams.rels_file, relf);
00058         } else {
```

```
00059             uParams.rels_file = NULL;
00060         }
00061
00062         uParams.eval_points = evpts;
00063         uParams.rank_aggregation_method = 200;
00064
00065         FLAGR_DRIVER(uParams);
00066
00067         if (uParams.input_file) free(uParams.input_file);
00068         if (uParams.rels_file) free(uParams.rels_file);
00069         if (uParams.rels_file) free(uParams.random_string);
00070         if (uParams.output_dir) free(uParams.output_dir);
00071     }
00072
00074     __declspec(dllexport) void __cdecl Copeland(const char inf[], const char relf[], const int evpts,
      const char ranstr[], const char out[]) {
00075
00076         struct UserParams uParams;
00077         srand(time(0));
00078
00079         uParams.input_file = (char *)malloc((strlen(inf) + 1) * sizeof(char));
00080         strcpy(uParams.input_file, inf);
00081
00082         uParams.random_string = (char *)malloc((strlen(ranstr) + 1) * sizeof(char));
00083         strcpy(uParams.random_string, ranstr);
00084
00085         uParams.output_dir = (char *)malloc((strlen(out) + 1) * sizeof(char));
00086         strcpy(uParams.output_dir, out);
00087
00088         if (strlen(relf) > 0) {
00089             uParams.rels_file = (char *)malloc((strlen(relf) + 1) * sizeof(char));
00090             strcpy(uParams.rels_file, relf);
00091         } else {
00092             uParams.rels_file = NULL;
00093         }
00094
00095         uParams.eval_points = evpts;
00096         uParams.rank_aggregation_method = 201;
00097
00098         FLAGR_DRIVER(uParams);
00099
00100         if (uParams.input_file) free(uParams.input_file);
00101         if (uParams.rels_file) free(uParams.rels_file);
00102         if (uParams.rels_file) free(uParams.random_string);
00103         if (uParams.output_dir) free(uParams.output_dir);
00104     }
00105
00107     __declspec(dllexport) void __cdecl OutrankingApproach
00108         (const char inf[], const char relf[], const int evpts, const char ranstr[], const char out[],
00109             const float pref_t, const float veto_t, const float conc_t, const float disc_t) {
00110
00111                 struct UserParams uParams;
00112                 srand(time(0));
00113
00114                 uParams.input_file = (char *)malloc((strlen(inf) + 1) * sizeof(char));
00115                 strcpy(uParams.input_file, inf);
00116
00117                 uParams.random_string = (char *)malloc((strlen(ranstr) + 1) * sizeof(char));
00118                 strcpy(uParams.random_string, ranstr);
00119
00120                 uParams.output_dir = (char *)malloc((strlen(out) + 1) * sizeof(char));
00121                 strcpy(uParams.output_dir, out);
00122
00123                 if (strlen(relf) > 0) {
00124                     uParams.rels_file = (char *)malloc((strlen(relf) + 1) * sizeof(char));
00125                     strcpy(uParams.rels_file, relf);
00126                 } else {
00127                     uParams.rels_file = NULL;
00128                 }
00129
00130                 uParams.eval_points = evpts;
00131                 uParams.rank_aggregation_method = 300;
00132                 uParams.pref_thr = pref_t;
00133                 uParams.veto_thr = veto_t;
00134                 uParams.conc_thr = conc_t;
00135                 uParams.disc_thr = disc_t;
00136
00137                 FLAGR_DRIVER(uParams);
00138
00139                 if (uParams.input_file) free(uParams.input_file);
00140                 if (uParams.rels_file) free(uParams.rels_file);
00141                 if (uParams.rels_file) free(uParams.random_string);
00142                 if (uParams.output_dir) free(uParams.output_dir);
00143     }
00144
00146     __declspec(dllexport) void __cdecl Kemeny(const char inf[], const char relf[], const int evpts,
      const char ranstr[], const char out[]) {
```

```
00147
00148         struct UserParams uParams;
00149         srand(time(0));
00150
00151         uParams.input_file = (char *)malloc((strlen(inf) + 1) * sizeof(char));
00152         strcpy(uParams.input_file, inf);
00153
00154         uParams.random_string = (char *)malloc((strlen(ranstr) + 1) * sizeof(char));
00155         strcpy(uParams.random_string, ranstr);
00156
00157         uParams.output_dir = (char *)malloc((strlen(out) + 1) * sizeof(char));
00158         strcpy(uParams.output_dir, out);
00159
00160         if (strlen(relf) > 0) {
00161             uParams.rels_file = (char *)malloc((strlen(relf) + 1) * sizeof(char));
00162             strcpy(uParams.rels_file, relf);
00163         } else {
00164             uParams.rels_file = NULL;
00165         }
00166
00167         uParams.eval_points = evpts;
00168         uParams.rank_aggregation_method = 400;
00169
00170         FLAGR_DRIVER(uParams);
00171
00172         if (uParams.input_file) free(uParams.input_file);
00173         if (uParams.rels_file) free(uParams.rels_file);
00174         if (uParams.rels_file) free(uParams.random_string);
00175         if (uParams.output_dir) free(uParams.output_dir);
00176     }
00177
00179     __declspec(dllexport) void __cdecl RobustRA
00180         (const char inf[], const char relf[], const int evpts, const char ranstr[], const char out[],
00181         const bool exact) {
00182
00183             struct UserParams uParams;
00184             srand(time(0));
00185
00186             uParams.input_file = (char *)malloc((strlen(inf) + 1) * sizeof(char));
00187             strcpy(uParams.input_file, inf);
00188
00189             uParams.random_string = (char *)malloc((strlen(ranstr) + 1) * sizeof(char));
00190             strcpy(uParams.random_string, ranstr);
00191
00192             uParams.output_dir = (char *)malloc((strlen(out) + 1) * sizeof(char));
00193             strcpy(uParams.output_dir, out);
00194
00195             if (strlen(relf) > 0) {
00196                 uParams.rels_file = (char *)malloc((strlen(relf) + 1) * sizeof(char));
00197                 strcpy(uParams.rels_file, relf);
00198             } else {
00199                 uParams.rels_file = NULL;
00200             }
00201
00202             uParams.eval_points = evpts;
00203             uParams.exact = exact;
00204             uParams.rank_aggregation_method = 401;
00205
00206             FLAGR_DRIVER(uParams);
00207
00208             if (uParams.input_file) free(uParams.input_file);
00209             if (uParams.rels_file) free(uParams.rels_file);
00210             if (uParams.rels_file) free(uParams.random_string);
00211             if (uParams.output_dir) free(uParams.output_dir);
00212     }
00213
00215     __declspec(dllexport) void __cdecl DIBRA
00216         (const char inf[], const char relf[], const int evpts, const int agg, const char ranstr[],
00217             const char out[], const int wnorm, const int dist, const bool prune, const float gamma,
00218             const float d1, const float d2, const float tol, const int iter,
00219             const float pref_t, const float veto_t, const float conc_t, const float disc_t) {
00220
00221                 struct UserParams uParams;
00222                 srand(time(0));
00223
00224                 uParams.input_file = (char *)malloc((strlen(inf) + 1) * sizeof(char));
00225                 strcpy(uParams.input_file, inf);
00226
00227                 uParams.random_string = (char *)malloc((strlen(ranstr) + 1) * sizeof(char));
00228                 strcpy(uParams.random_string, ranstr);
00229
00230                 uParams.output_dir = (char *)malloc((strlen(out) + 1) * sizeof(char));
00231                 strcpy(uParams.output_dir, out);
00232
00233                 if (strlen(relf) > 0) {
00234                     uParams.rels_file = (char *)malloc((strlen(relf) + 1) * sizeof(char));
00235                     strcpy(uParams.rels_file, relf);
```

```
00236                 } else {
00237                     uParams.rels_file = NULL;
00238                 }
00239 
00240                 uParams.eval_points = evpts;
00241                 uParams.rank_aggregation_method = agg;
00242                 uParams.weight_normalization = wnorm;
00243                 uParams.distance = dist;
00244                 uParams.prune = prune;
00245                 uParams.gamma = gamma;
00246                 uParams.delta1 = d1;
00247                 uParams.delta2 = d2;
00248                 uParams.tol = tol;
00249                 uParams.max_iter = iter;
00250                 uParams.pref_thr = pref_t;
00251                 uParams.veto_thr = veto_t;
00252                 uParams.conc_thr = conc_t;
00253                 uParams.disc_thr = disc_t;
00254 
00255                 FLAGR_DRIVER(uParams);
00256 
00257                 if (uParams.input_file) free(uParams.input_file);
00258                 if (uParams.rels_file) free(uParams.rels_file);
00259                 if (uParams.rels_file) free(uParams.random_string);
00260                 if (uParams.output_dir) free(uParams.output_dir);
00261     }
00262 
00264     __declspec(dllexport) void __cdecl PrefRel(const char inf[], const char relf[], const int evpts,
      const char ranstr[],
00265                 const char out[], const float alpha, const float beta) {
00266 
00267                 struct UserParams uParams;
00268                 srand(time(0));
00269 
00270                 uParams.input_file = (char *)malloc((strlen(inf) + 1) * sizeof(char));
00271                 strcpy(uParams.input_file, inf);
00272 
00273                 uParams.random_string = (char *)malloc((strlen(ranstr) + 1) * sizeof(char));
00274                 strcpy(uParams.random_string, ranstr);
00275 
00276                 uParams.output_dir = (char *)malloc((strlen(out) + 1) * sizeof(char));
00277                 strcpy(uParams.output_dir, out);
00278 
00279                 if (strlen(relf) > 0) {
00280                     uParams.rels_file = (char *)malloc((strlen(relf) + 1) * sizeof(char));
00281                     strcpy(uParams.rels_file, relf);
00282                 } else {
00283                     uParams.rels_file = NULL;
00284                 }
00285 
00286                 uParams.eval_points = evpts;
00287                 uParams.rank_aggregation_method = 600;
00288                 uParams.alpha = alpha;
00289                 uParams.beta = beta;
00290 
00291                 FLAGR_DRIVER(uParams);
00292 
00293                 if (uParams.input_file) free(uParams.input_file);
00294                 if (uParams.rels_file) free(uParams.rels_file);
00295                 if (uParams.rels_file) free(uParams.random_string);
00296                 if (uParams.output_dir) free(uParams.output_dir);
00297     }
00298 
00300     __declspec(dllexport) void __cdecl Agglomerative(const char inf[], const char relf[], const int
      evpts, const char ranstr[],
00301                 const char out[], const float c1, const float c2) {
00302 
00303                 struct UserParams uParams;
00304                 srand(time(0));
00305 
00306                 uParams.input_file = (char *)malloc((strlen(inf) + 1) * sizeof(char));
00307                 strcpy(uParams.input_file, inf);
00308 
00309                 uParams.random_string = (char *)malloc((strlen(ranstr) + 1) * sizeof(char));
00310                 strcpy(uParams.random_string, ranstr);
00311 
00312                 uParams.output_dir = (char *)malloc((strlen(out) + 1) * sizeof(char));
00313                 strcpy(uParams.output_dir, out);
00314 
00315                 if (strlen(relf) > 0) {
00316                     uParams.rels_file = (char *)malloc((strlen(relf) + 1) * sizeof(char));
00317                     strcpy(uParams.rels_file, relf);
00318                 } else {
00319                     uParams.rels_file = NULL;
00320                 }
00321 
00322                 uParams.eval_points = evpts;
```

```
00323                 uParams.rank_aggregation_method = 700;
00324                 uParams.c1 = c1;
00325                 uParams.c2 = c2;
00326 
00327                 FLAGR_DRIVER(uParams);
00328 
00329                 if (uParams.input_file) free(uParams.input_file);
00330                 if (uParams.rels_file) free(uParams.rels_file);
00331                 if (uParams.rels_file) free(uParams.random_string);
00332                 if (uParams.output_dir) free(uParams.output_dir);
00333     }
00334 
00336     __declspec(dllexport) void __cdecl MC(const char inf[], const char relf[], const int evpts, const
      int ram, const char ranstr[],
00337             const char out[], const float ergodic_number, const float delta, const int iter) {
00338 
00339                 struct UserParams uParams;
00340                 srand(time(0));
00341 
00342                 uParams.input_file = (char *)malloc((strlen(inf) + 1) * sizeof(char));
00343                 strcpy(uParams.input_file, inf);
00344 
00345                 uParams.random_string = (char *)malloc((strlen(ranstr) + 1) * sizeof(char));
00346                 strcpy(uParams.random_string, ranstr);
00347 
00348                 uParams.output_dir = (char *)malloc((strlen(out) + 1) * sizeof(char));
00349                 strcpy(uParams.output_dir, out);
00350 
00351                 if (strlen(relf) > 0) {
00352                     uParams.rels_file = (char *)malloc((strlen(relf) + 1) * sizeof(char));
00353                     strcpy(uParams.rels_file, relf);
00354                 } else {
00355                     uParams.rels_file = NULL;
00356                 }
00357 
00358                 uParams.eval_points = evpts;
00359                 uParams.rank_aggregation_method = ram;
00360                 uParams.alpha = ergodic_number;
00361                 uParams.delta1 = delta;
00362                 uParams.max_iter = iter;
00363 
00364                 FLAGR_DRIVER(uParams);
00365 
00366                 if (uParams.input_file) free(uParams.input_file);
00367                 if (uParams.rels_file) free(uParams.rels_file);
00368                 if (uParams.rels_file) free(uParams.random_string);
00369                 if (uParams.output_dir) free(uParams.output_dir);
00370     }
00371 
00373     __declspec(dllexport) void __cdecl Custom1(const char inf[], const char relf[], const int evpts,
    const char ranstr[], const char out[]) {
00374 
00375         struct UserParams uParams;
00376         srand(time(0));
00377 
00378         uParams.input_file = (char *)malloc((strlen(inf) + 1) * sizeof(char));
00379         strcpy(uParams.input_file, inf);
00380 
00381         uParams.random_string = (char *)malloc((strlen(ranstr) + 1) * sizeof(char));
00382         strcpy(uParams.random_string, ranstr);
00383 
00384         uParams.output_dir = (char *)malloc((strlen(out) + 1) * sizeof(char));
00385         strcpy(uParams.output_dir, out);
00386 
00387         if (strlen(relf) > 0) {
00388             uParams.rels_file = (char *)malloc((strlen(relf) + 1) * sizeof(char));
00389             strcpy(uParams.rels_file, relf);
00390         } else {
00391             uParams.rels_file = NULL;
00392         }
00393 
00394         uParams.eval_points = evpts;
00395         uParams.rank_aggregation_method = 901;
00396 
00397         FLAGR_DRIVER(uParams);
00398 
00399         if (uParams.input_file) free(uParams.input_file);
00400         if (uParams.rels_file) free(uParams.rels_file);
00401         if (uParams.rels_file) free(uParams.random_string);
00402         if (uParams.output_dir) free(uParams.output_dir);
00403     }
00404 
00406     __declspec(dllexport) void __cdecl Custom2(const char inf[], const char relf[], const int evpts,
    const char ranstr[], const char out[]) {
00407 
00408         struct UserParams uParams;
00409         srand(time(0));
```

```
00410
00411         uParams.input_file = (char *)malloc((strlen(inf) + 1) * sizeof(char));
00412         strcpy(uParams.input_file, inf);
00413
00414         uParams.random_string = (char *)malloc((strlen(ranstr) + 1) * sizeof(char));
00415         strcpy(uParams.random_string, ranstr);
00416
00417         uParams.output_dir = (char *)malloc((strlen(out) + 1) * sizeof(char));
00418         strcpy(uParams.output_dir, out);
00419
00420         if (strlen(relf) > 0) {
00421             uParams.rels_file = (char *)malloc((strlen(relf) + 1) * sizeof(char));
00422             strcpy(uParams.rels_file, relf);
00423         } else {
00424             uParams.rels_file = NULL;
00425         }
00426
00427         uParams.eval_points = evpts;
00428         uParams.rank_aggregation_method = 902;
00429
00430         FLAGR_DRIVER(uParams);
00431
00432         if (uParams.input_file) free(uParams.input_file);
00433         if (uParams.rels_file) free(uParams.rels_file);
00434         if (uParams.rels_file) free(uParams.random_string);
00435         if (uParams.output_dir) free(uParams.output_dir);
00436     }
00437
00438 }
```

6.5 FLAGR/driver.cpp File Reference

```
#include "stdio.h"
#include "stdlib.h"
#include <stdint.h>
#include <string.h>
#include <math.h>
#include <time.h>
#include <dirent.h>
#include <assert.h>
#include <vector>
#include "src/SimpleScoreStats.cpp"
#include "src/Rel.cpp"
#include "src/Rels.cpp"
#include "src/InputParams.cpp"
#include "src/Voter.cpp"
#include "src/InputItem.cpp"
#include "src/InputList.cpp"
#include "src/Ranking.cpp"
#include "src/MergedItem.cpp"
#include "src/MergedList.cpp"
#include "src/Evaluator.cpp"
#include "src/Aggregator.cpp"
#include "src/Query.cpp"
#include "src/input/InputData.cpp"
```

Macros

- #define MAX_LIST_ITEMS 1000
- #define NOT_RANKED_ITEM_RANK MAX_LIST_ITEMS * MAX_LIST_ITEMS

Typedefs

- typedef uint32_t rank_t
- typedef double score_t

Functions

- void FLAGR_DRIVER (UserParams uParams)

  This function drives the entire FLAGR execution - It passes the Python Parameters to FLAGR.

6.5.1 Macro Definition Documentation

6.5.1.1 MAX_LIST_ITEMS

```
#define MAX_LIST_ITEMS 1000
```

Definition at line 43 of file driver.cpp.

6.5.1.2 NOT_RANKED_ITEM_RANK

```
#define NOT_RANKED_ITEM_RANK MAX_LIST_ITEMS * MAX_LIST_ITEMS
```

Definition at line 44 of file driver.cpp.

6.5.2 Typedef Documentation

6.5.2.1 rank_t

```
typedef uint32_t rank_t
```

FLAGR - Fuse, Learn, Aggregate, Rerank A high performance library for rank aggregation

References: [1] E. Renda, U. Straccia, "Web metasearch: rank vs. score based rank aggregation methods", In Proceedings of the 2003 ACM symposium on Applied computing, pp. 841-846, 2003. [2] M. Farah, D. Vanderpooten, "An outranking approach for rank aggregation in information retrieval", In Proceedings of the 30th ACM Conference on Research and Development in Information Retrieval, pp. 591-598, 2007. [3] M.S. Desarkar, S. Sarkar, P. Mitra, "Preference relations based unsupervised rank aggregation for metasearch", Expert Systems with Applications, vol. 49, pp. 86-98, 2016. [4] S. Chatterjee, A. Mukhopadhyay, M. Bhattacharyya, "A weighted rank aggregation approach towards crowd opinion analysis", Knowledge-Based Systems, vol. 149, pp. 47-60, 2018. [5] L. Akritidis, A. Fevgas, P. Bozanis, Y. Manolopoulos, "An Unsupervised Distance-Based Model for Weighted Rank Aggregation with List Pruning", Expert Systems with Applications, vol. 202, pp. 117435, 2022. [6] C. Dwork, R. Kumar, M. Naor, D. Sivakumar, "Rank Aggregation Methods for the Web", In Proceedings of the 10th International Conference on World Wide Web, pp. 613-622, 2001. [7] R. Kolde, S. Laur, P. Adler, J. Vilo, "Robust rank aggregation for gene list integration and meta-analysis", Bioinformatics, vol. 28, no. 4, pp. 573–580, 2012. [8] K.L. Majumder, G.P. Bhattacharjee, "Algorithm AS 63: The incomplete Beta Integral", Applied Statistics, vol. 22, no. 3, pp. 409-411, 1973. [9] R.P. DeConde, S. Hawley, S. Falcon, N. Clegg, B. Knudsen, R. Etzioni, "Combining results of microarray experiments: a rank aggregation approach", Statistical Applications in Genetics and Molecular Biology, vol. 5, no. 1, 2006.

Definition at line 40 of file driver.cpp.

6.5.2.2 score_t

```
typedef double score_t
```

Definition at line 41 of file driver.cpp.

6.5.3 Function Documentation

6.5.3.1 FLAGR_DRIVER()

```
void FLAGR_DRIVER (
            UserParams uParams )
```

This function drives the entire FLAGR execution - It passes the Python Parameters to FLAGR.

Definition at line 63 of file driver.cpp.

6.6 driver.cpp

Go to the documentation of this file.

```
00001
00029 #include "stdio.h"
00030 #include "stdlib.h"
00031 #include <stdint.h>
00032 #include <string.h>
00033 #include <math.h>
00034 #include <time.h>
00035 #include <dirent.h>
00036 #include <assert.h>
00037 #include <string.h>
00038 #include <vector>
00039
00040 typedef uint32_t rank_t;
00041 typedef double score_t;
00042
00043 #define MAX_LIST_ITEMS 1000
00044 #define NOT_RANKED_ITEM_RANK MAX_LIST_ITEMS * MAX_LIST_ITEMS
00045
00046 #include "src/SimpleScoreStats.cpp"
00047 #include "src/Rel.cpp"
00048 #include "src/Rels.cpp"
00049 #include "src/InputParams.cpp"
00050 #include "src/Voter.cpp"
00051 #include "src/InputItem.cpp"
00052 #include "src/InputList.cpp"
00053 #include "src/Ranking.cpp"
00054 #include "src/MergedItem.cpp"
00055 #include "src/MergedList.cpp"
00056 #include "src/Evaluator.cpp"
00057 #include "src/Aggregator.cpp"
00058 #include "src/Query.cpp"
00059
00060 #include "src/input/InputData.cpp"
00061
00063 void FLAGR_DRIVER (UserParams uParams) {
00064     class InputParams * PARAMS = new InputParams(uParams);
00065 //  PARAMS->display(); fflush(NULL);
00066
00067     class InputData * input_data = new InputData(PARAMS);
00068
00069     input_data->aggregate();
00070
00071     if (uParams.rels_file) {
00072         input_data->evaluate();
00073     }
00074
00075     delete input_data;
00076     delete PARAMS;
00077 }
```

6.7 FLAGR/main.cpp File Reference

```
#include "cflagr.cpp"
```

Functions

- int main (int argc, char ∗argv[ ])

6.7.1 Function Documentation

6.7.1.1 main()

```
int main (
            int argc,
            char * argv[] )
```

Main Function Input arguments: 0: program name 1: Evaluation cut off point. How many elements of the aggregate list will be included in the evaluation process. If nothing is passed, the default value is 10. 2: Input file: This where the aggregation algorithms read data from. If nothing is passed, the default value is ../../examples/testdata/testdata.csv. 3: Output directory: This is where the program writes the generated aggregate lists and the results of the evaluation process. If nothing is passed, the default value is output. 4: Qrels file: This is where the program reads the relevance judgments of the list elements. This file is required for the evaluation to take place. If nothing is passed, the default value is ../../examples/testdata/testdata_qrels.csv.

Definition at line 17 of file main.cpp.

6.8 main.cpp

Go to the documentation of this file.

```
00001 #include "cflagr.cpp"
00002
00017 int main(int argc, char *argv[]) {
00018     char * input_file = NULL, *qrels_file = NULL, *output_dir = NULL;
00019     int cut_off = 0;
00020
00021     if (argc == 1) {
00022         input_file = new char[strlen("../../examples/testdata/testdata.csv") + 1];
00023         strcpy(input_file, "../../examples/testdata/testdata.csv");
00024
00025         qrels_file = new char[strlen("../../examples/testdata/testdata_qrels.csv") + 1];
00026         strcpy(qrels_file, "../../examples/testdata/testdata_qrels.csv");
00027
00028         output_dir = new char[strlen("output") + 1];
00029         strcpy(output_dir, "output");
00030
00031         cut_off = 10;
00032
00033     } else if (argc == 2) {
00034         input_file = new char[strlen("../../examples/testdata/testdata.csv") + 1];
00035         strcpy(input_file, "../../examples/testdata/testdata.csv");
00036
00037         qrels_file = new char[strlen("../../examples/testdata/testdata_qrels.csv") + 1];
00038         strcpy(qrels_file, "../../examples/testdata/testdata_qrels.csv");
00039
00040         output_dir = new char[strlen("output") + 1];
00041         strcpy(output_dir, "output");
```

```
00042
00043         cut_off = atoi(argv[1]);
00044
00045     } else if (argc == 3) {
00046         input_file = new char[strlen(argv[2]) + 1];
00047         strcpy(input_file, argv[2]);
00048
00049         qrels_file = new char[1];
00050         strcpy(qrels_file, "");
00051
00052         output_dir = new char[strlen("output") + 1];
00053         strcpy(output_dir, "output");
00054
00055         cut_off = atoi(argv[1]);
00056
00057     } else if (argc == 4) {
00058         input_file = new char[strlen(argv[2]) + 1];
00059         strcpy(input_file, argv[2]);
00060
00061         qrels_file = new char[1];
00062         strcpy(qrels_file, "");
00063
00064         output_dir = new char[strlen(argv[3]) + 1];
00065         strcpy(output_dir, argv[3]);
00066
00067         cut_off = atoi(argv[1]);
00068
00069     } else if (argc == 5) {
00070         input_file = new char[strlen(argv[2]) + 1];
00071         strcpy(input_file, argv[2]);
00072
00073         qrels_file = new char[strlen(argv[4]) + 1];
00074         strcpy(qrels_file, argv[4]);
00075
00076         output_dir = new char[strlen(argv[3]) + 1];
00077         strcpy(output_dir, argv[3]);
00078
00079         cut_off = atoi(argv[1]);
00080     }
00081
00082     printf("Aggregating with linear combination (rank normalization)...\n"); fflush(NULL);
00083     Linear(input_file, qrels_file, cut_off, 101, "Linear_out", output_dir);
00084
00085     printf("Aggregating with Borda Count...\n"); fflush(NULL);
00086     Linear(input_file, qrels_file, cut_off, 100, "Borda_out", output_dir);
00087
00088     printf("Aggregating with Condorcet Winners...\n"); fflush(NULL);
00089     Condorcet(input_file, qrels_file, cut_off, "Condorcet_out", output_dir);
00090
00091     printf("Aggregating with Copeland Winners...\n"); fflush(NULL);
00092     Copeland(input_file, qrels_file, cut_off, "Copeland_out", output_dir);
00093
00094     printf("Aggregating with the Outranking Approach...\n"); fflush(NULL);
00095     OutrankingApproach(input_file, qrels_file, cut_off, "Outrank_out", output_dir, 0.0, 0.75, 0.0,
    0.25);
00096
00097 //  Kemeny(input_file, qrels_file, 20, "Kemeny_out", output_dir);
00098
00099     printf("Aggregating with Markov Chains method 4...\n"); fflush(NULL);
00100     MC(input_file, qrels_file, cut_off, 804, "MC_out", output_dir, 0.15, 0.00000001, 200);
00101
00102     printf("Aggregating with Robust Rank Aggregation...\n"); fflush(NULL);
00103     RobustRA(input_file, qrels_file, cut_off, "RRA", output_dir, false);
00104
00105     printf("Aggregating with the Preference Relations Method...\n"); fflush(NULL);
00106     PrefRel(input_file, qrels_file, cut_off, "PrefRel_out", output_dir, 0.3, 0.2);
00107
00108     printf("Aggregating with the Agglomerative Method...\n"); fflush(NULL);
00109     Agglomerative(input_file, qrels_file, cut_off, "Agglomerative_out", output_dir, 2.5, 1.5);
00110
00111     printf("Aggregating with DIBRA + Outranking Approach...\n"); fflush(NULL);
00112     DIBRA(input_file, qrels_file, cut_off, 5300, "DIBRA_out", output_dir, 2, 3, false, 1.5, 0.4, 0.1,
    0.01, 50, 0.0, 0.75, 0.0, 0.25);
00113
00114     delete [] input_file;
00115     delete [] qrels_file;
00116     delete [] output_dir;
00117
00118     printf("\n\nDone.  Press any key to exit...\n"); fflush(NULL);
00119     getchar();
00120     return 0;
00121 }
00122
```

6.9 FLAGR/README.md File Reference

6.10 FLAGR/src/Aggregator.cpp File Reference

```
#include "Aggregator.h"
```

6.11 Aggregator.cpp

Go to the documentation of this file.

```
00001 #include "Aggregator.h"
00002
00004 Aggregator::Aggregator() :
00005         num_lists(0),
00006         num_alloc_lists(4),
00007         input_lists((class InputList **)malloc(this->num_alloc_lists * sizeof(class InputList *))),
00008         output_list(NULL) { }
00009
00011 Aggregator::~Aggregator() {
00012     if (this->input_lists) {
00013         for (uint16_t i = 0; i < this->num_lists; i++) {
00014             if (this->input_lists[i]) {
00015                 delete this->input_lists[i];
00016             }
00017         }
00018         free(this->input_lists);
00019     }
00020
00021     if (this->output_list) {
00022         delete this->output_list;
00023         this->output_list = NULL;
00024     }
00025 }
00026
00028 class InputList * Aggregator::create_list(char * v, double w) {
00029     this->input_lists[this->num_lists] = new InputList(this->num_lists, v, w);
00030     this->num_lists++;
00031
00032     if (this->num_lists >= this->num_alloc_lists) {
00033         this->num_alloc_lists *= 2;
00034         this->input_lists = (class InputList **)realloc
00035                 (this->input_lists, this->num_alloc_lists * sizeof(class InputList *));
00036     }
00037
00038     return this->input_lists[this->num_lists - 1];
00039 }
00040
00042 void Aggregator::merge_input_lists() {
00043     uint32_t l = 0, k = 0;
00044
00045     this->output_list = new MergedList(1024, this->num_lists);
00046
00047     for (l = 0; l < this->num_lists; l++) {
00048         for (k = 0; k < this->input_lists[l]->get_num_items(); k++) {
00049             this->output_list->insert(this->input_lists[l]->get_item(k), l, this->input_lists);
00050         }
00051     }
00052
00053     this->output_list->convert_to_array();
00054 }
00055
00057 void Aggregator::init_weights() {
00058     for (uint32_t l = 0; l < this->num_lists; l++) {
00059         this->input_lists[l]->get_voter()->set_weight(1.0);
00060     }
00061 }
00062
00064 void Aggregator::destroy_output_list() {
00065     if (this->output_list) {
00066         delete this->output_list;
00067         this->output_list = NULL;
00068     }
00069 }
00070
00072 class Voter ** Aggregator::aggregate(char * topic, class InputParams * params) {
```

```
00073
00074     class Voter ** voters_list = NULL;
00075     uint32_t ram = params->get_aggregation_method();
00076
00077     class SimpleScoreStats s;
00078     s.set_min_val(0.0);
00079     s.set_max_val(1.0);
00080     s.set_mean_val(0.0);
00081     s.set_std_val(1.0);
00082
00085     if (ram >= 100 && ram <= 109) {
00086         this->merge_input_lists();
00087         this->output_list->CombSUM(this->input_lists, &s, params);
00088
00090     } else if (ram >= 110 && ram <= 119) {
00091         this->merge_input_lists();
00092         this->output_list->CombMNZ(this->input_lists, &s, params);
00093
00095     } else if (ram == 200) {
00096         this->merge_input_lists();
00097         this->output_list->CondorcetWinners(this->input_lists, &s, params);
00098
00100     } else if (ram == 201) {
00101         this->merge_input_lists();
00102         this->output_list->CopelandWinners(this->input_lists, &s, params);
00103
00105     } else if (ram == 300) {
00106         this->merge_input_lists();
00107         this->output_list->Outranking(this->input_lists, &s, params);
00108
00110     } else if (ram == 400) {
00111         this->merge_input_lists();
00112         this->output_list->KemenyOptimal(this->input_lists, &s, params);
00113
00115     } else if (ram == 401) {
00116         this->merge_input_lists();
00117         this->output_list->RobustRA(this->input_lists, &s, params);
00118
00120     } else if (ram >= 5100 && ram <= 5999) {
00121         this->merge_input_lists();
00122         voters_list = this->output_list->DIBRA(this->input_lists, &s, params);
00123
00125     } else if (ram == 600) {
00126         this->merge_input_lists();
00127         this->output_list->PrefRel(this->input_lists, &s, params);
00128
00130     } else if (ram == 700) {
00131         class MergedList * temp = new MergedList(1024, this->num_lists);
00132         this->output_list = temp->Agglomerative(this->input_lists, &s, params);
00133         delete temp;
00134
00136     } else if (ram == 801 || ram == 802 || ram == 803 || ram == 804 || ram == 805) {
00137         this->merge_input_lists();
00138         this->output_list->MC(this->input_lists, &s, params);
00139
00141     } else if (ram == 901) {
00142         this->merge_input_lists();
00143         this->output_list->CustomMethod1(this->input_lists, &s, params);
00144
00146     } else if (ram == 902) {
00147         this->merge_input_lists();
00148         this->output_list->CustomMethod2(this->input_lists, &s, params);
00149     }
00150
00151
00152
00153     this->output_list->write_to_CSV(topic, params);
00154 //  this->output_list->display_list(); getchar();
00155 //  printf("topic %s - ok\n", topic); fflush(NULL);
00156     return voters_list;
00157 }
00158
00159 void Aggregator::display() {
00160     for (uint32_t i = 0; i < this->num_lists; i++) {
00161         if (this->input_lists[i]) {
00162             printf("\t=== Displaying Input List %d:\n", i);
00163             this->input_lists[i]->display();
00164             getchar();
00165         }
00166     }
00167
00168     if (this->output_list) {
00169         this->output_list->display();
00170     }
00171 }
00172
00174 inline uint16_t Aggregator::get_num_lists() { return this->num_lists; }
```

```
00175 inline rank_t Aggregator::get_num_items() { return this->output_list->get_num_items(); }
00176 inline class InputList * Aggregator::get_input_list(uint32_t i) { return this->input_lists[i]; }
00177 inline class MergedList * Aggregator::get_output_list() { return this->output_list; }
```

6.12 FLAGR/src/Aggregator.h File Reference

Classes

- class Aggregator

6.13 Aggregator.h

Go to the documentation of this file.

```
00001 #ifndef AGGREGATOR_H
00002 #define AGGREGATOR_H
00003
00004 class Aggregator {
00005     private:
00006         uint16_t num_lists;
00007         uint16_t num_alloc_lists;
00008         class InputList ** input_lists;
00009         class MergedList * output_list;
00010
00011     private:
00012         void merge_input_lists();
00013
00014     public:
00015         Aggregator();
00016         ~Aggregator();
00017
00018         class InputList * create_list(char *, double);
00019         class Voter ** aggregate(char *, class InputParams *);
00020         void init_weights();
00021         void destroy_output_list();
00022         void display();
00023
00025         uint16_t get_num_lists();
00026         rank_t get_num_items();
00027         class InputList * get_input_list(uint32_t);
00028         class MergedList * get_output_list();
00029 };
00030
00031 #endif // AGGREGATOR_H
```

6.14 FLAGR/src/Evaluator.cpp File Reference

```
#include "Evaluator.h"
```

6.15 Evaluator.cpp

Go to the documentation of this file.

```
00001 #include "Evaluator.h"
00002
00004 Evaluator::Evaluator() :
00005         relevs (new Rels(1024)),
00006         true_positives(0),
00007         average_precision(0.0),
00008         average_recall(0.0),
00009         average_dcg(0.0),
00010         average_ndcg(0.0),
```

```
00011         precision(NULL),
00012         recall(NULL),
00013         dcg(NULL),
00014         ndcg(NULL) { }
00015
00017 Evaluator::~Evaluator() {
00018     if (this->relevs) {
00019         delete this->relevs;
00020         this->relevs = NULL;
00021     }
00022
00023     this->clear();
00024 }
00025
00027 void Evaluator::insert_relev(char * r, uint32_t j) {
00028     this->relevs->insert(r, j);
00029 }
00030
00031
00032 void Evaluator::clear() {
00033     if (this->precision) {
00034         delete [] this->precision;
00035         this->precision = NULL;
00036     }
00037
00038     if (this->recall) {
00039         delete [] this->recall;
00040         this->recall = NULL;
00041     }
00042
00043     if (this->dcg) {
00044         delete [] this->dcg;
00045         this->dcg = NULL;
00046     }
00047
00048     if (this->ndcg) {
00049         delete [] this->ndcg;
00050         this->ndcg = NULL;
00051     }
00052 }
00053
00055 void Evaluator::evaluate(rank_t ev_pts, char * qry, class MergedList * lst, FILE * eval_file) {
00056     rank_t num_items = lst->get_num_items();
00057     rank_t num_relevant_items = this->relevs->get_num_nodes();
00058     uint32_t relevance = 0;
00059     double qty = 0.0;
00060
00061 //  this->relevs->display(); getchar();
00062
00063     this->true_positives = 0;
00064     this->average_precision = 0.0;
00065     this->average_recall = 0.0;
00066     this->average_dcg = 0.0;
00067     this->average_ndcg = 0.0;
00068     this->precision = new double[num_items];
00069     this->recall = new double[num_items];
00070     this->dcg = new double[num_items];
00071     this->ndcg = new double[num_items];
00072
00074     double * ideal_dcg = new double[num_items];
00075
00076     for (rank_t i = 0; i < num_items; i++) {
00077         class MergedItem * p = lst->get_item(i);
00078 //      printf("Checking %d - %s:  \n", i + 1, p->get_code()); fflush(NULL);
00079         if (this->relevs->search(p->get_code(), &relevance)) {
00080             if (p->get_final_score() > 0 && relevance > 0) {
00081                 this->true_positives++;
00082                 this->average_precision += this->true_positives / (i + 1.0);
00083                 if (num_relevant_items > 0) {
00084                     this->average_recall += this->true_positives / (double) num_relevant_items;
00085                 }
00086             }
00087 //          printf(" * (%d) == METRICS : \n", relevance);
00088         }
00089
00091         this->precision[i] = this->true_positives / (i + 1.0);
00092
00094         if (num_relevant_items > 0) {
00095             this->recall[i] = this->true_positives / (double) num_relevant_items;
00096         } else {
00097             this->recall[i] = 0;
00098         }
00099
00101         if (i == 0) {
00102 //          this->dcg[i] = relevance / log2(i + 2.0);  // Version 1
00103             this->dcg[i] = (pow(2.0, relevance) - 1.0) / log2(i + 2.0);  // Version 2
00104         } else {
```

```
00105 //          this->dcg[i] = this->dcg[i - 1] + relevance / log2(i + 2.0);  // Version 1
00106             this->dcg[i] = this->dcg[i - 1] + (pow(2.0, relevance) - 1.0) / log2(i + 2.0);  // Version
    2
00107         }
00108
00110         ideal_dcg[i] = (double)relevance;
00111
00112         if (p->get_final_score() > 0 && relevance > 0) {
00113             this->average_dcg += this->dcg[i];
00114             this->average_ndcg += this->dcg[i] / (i + 2.0);
00115         }
00116 //      printf("Relevance:  %d - P@%d=%5.3f - R@%d=%5.3f - DCG@%d=%5.3f\n",
00117 //          relevance, i+1, this->precision[i], i+1, this->recall[i], i+1, this->dcg[i]); getchar();
00118     }
00119
00121     qsort(ideal_dcg, num_items, sizeof(double), &Evaluator::cmp_dbl);
00122
00124     for (rank_t i = 0; i < num_items; i++) {
00125         qty += (pow(2.00, ideal_dcg[i]) - 1.0) / (log2(i + 2.0));
00126 //      qty += ideal_dcg[i] / (log2(i + 2.0));
00127 //      printf("%2.1f\n", ideal_dcg[i]);
00128         if (qty > 0) {
00129             this->ndcg[i] = this->dcg[i] / qty;
00130         } else {
00131             this->ndcg[i] = 0.0;
00132         }
00133     }
00134
00136     if (this->true_positives > 0) {
00137         this->average_precision = this->average_precision / (double)num_relevant_items;
00138         this->average_recall = this->average_recall / (double)num_relevant_items;
00139         this->average_dcg = this->average_dcg / (double)num_relevant_items;
00140         this->average_ndcg = this->average_ndcg / (double)num_relevant_items;
00141     } else {
00142         this->average_precision = 0.0;
00143         this->average_recall = 0.0;
00144         this->average_dcg = 0.0;
00145         this->average_ndcg = 0.0;
00146     }
00147
00148     delete [] ideal_dcg;
00149
00150     char str[100];
00151     sprintf(str, "Topic %s", qry);
00152
00154     if (eval_file) {
00156         fprintf(eval_file, "%s,", str);
00157         fprintf(eval_file, "%d,", num_items);
00158         fprintf(eval_file, "%d,", num_relevant_items);
00159         fprintf(eval_file, "%d,", this->true_positives);
00160
00161         fprintf(eval_file, "%7.6f,", this->average_precision);
00162 //      fprintf(eval_file, "%7.6f,", this->average_recall);
00163 //      fprintf(eval_file, "%7.6f,", this->average_dcg);
00164 //      fprintf(eval_file, "%7.6f,", this->average_ndcg);
00165
00166         for (rank_t i = 0; i < ev_pts; i++) {
00167             fprintf(eval_file, "%7.6f,", num_items > i ?  this->precision[i] :  0);
00168         }
00169
00170         for (rank_t i = 0; i < ev_pts; i++) {
00171             fprintf(eval_file, "%7.6f,", num_items > i ?  this->recall[i] :  0);
00172         }
00173
00174         for (rank_t i = 0; i < ev_pts; i++) {
00175             fprintf(eval_file, "%7.6f,", num_items > i ?  this->dcg[i] :  0);
00176         }
00177
00178         for (rank_t i = 0; i < ev_pts; i++) {
00179             if (i < ev_pts - 1) {
00180                 fprintf(eval_file, "%7.6f,", num_items > i ?  this->ndcg[i] :  0);
00181             } else {
00182                 fprintf(eval_file, "%7.6f", num_items > i ?  this->ndcg[i] :  0);
00183             }
00184         }
00185         fprintf(eval_file, "\n");
00186     }
00187 }
00188
00190 double Evaluator::evaluate_input(class InputList * lst) {
00191     rank_t num_items = lst->get_num_items();
00192     rank_t num_relevant_items = this->relevs->get_num_nodes();
00193     uint32_t relevance = 0;
00194     double qty = 0.0;
00195
00196     this->true_positives = 0;
00197     this->average_precision = 0.0;
```

```
00198     this->average_recall = 0.0;
00199     this->average_dcg = 0.0;
00200     this->average_ndcg = 0.0;
00201     this->precision = new double[num_items];
00202     this->recall = new double[num_items];
00203     this->dcg = new double[num_items];
00204     this->ndcg = new double[num_items];
00205 
00207     double * ideal_dcg = new double[num_items];
00208 
00209     for (rank_t i = 0; i < num_items; i++) {
00210         class InputItem * p = lst->get_item(i);
00211 
00212 //      if (strcmp(lst->get_voter()->get_name(), "input.srchvrs11b.gz") == 0 && tid == 145) {
00213 //          printf("Checking %d - %s:  ", i + 1, p->get_code()); fflush(NULL);
00214 //          getchar();
00215 //      }
00216         if (this->relevs->search(p->get_code(), &relevance)) {
00217             this->true_positives++;
00218             this->average_precision += this->true_positives / (i + 1.0);
00219             if (num_relevant_items > 0) {
00220                 this->average_recall += this->true_positives / (double) num_relevant_items;
00221             }
00222 //          printf(" * (%d) == METRICS : ", relevance);
00223         }
00224 
00226         this->precision[i] = this->true_positives / (i + 1.0);
00227 
00229         if (num_relevant_items > 0) {
00230             this->recall[i] = this->true_positives / (double) num_relevant_items;
00231         } else {
00232             this->recall[i] = 0;
00233         }
00234 
00236         if (i == 0) {
00237             this->dcg[i] = (pow(2.0, relevance) - 1.0) / log2(i + 2.0);
00238 //          this->dcg[i] = relevance / log2(i + 2.0);
00239         } else {
00240             this->dcg[i] = this->dcg[i - 1] + (pow(2.0, relevance) - 1.0) / log2(i + 2.0);
00241 //          this->dcg[i] = this->dcg[i - 1] + relevance / log2(i + 2.0);
00242         }
00243 
00245         ideal_dcg[i] = (double)relevance;
00246 
00247         if (relevance > 0) {
00248             this->average_dcg += this->dcg[i];
00249             this->average_ndcg += this->dcg[i] / (i + 1.0);
00250         }
00251 //      printf("P@%d=%5.3f - R@%d=%5.3f - DCG@%d=%5.3f\n",
00252 //          i+1, this->precision[i], i+1, this->recall[i], i+1, this->dcg[i]);
00253     }
00254 
00256     qsort(ideal_dcg, num_items, sizeof(double), &Evaluator::cmp_dbl);
00257 
00259     for (rank_t i = 0; i < num_items; i++) {
00260         qty += (pow(2.00, ideal_dcg[i]) - 1.0) / (log2(i + 2.0));
00261 //      qty += ideal_dcg[i] / (log2(i + 2.0));
00262         if (qty > 0) {
00263             this->ndcg[i] = this->dcg[i] / qty;
00264         } else {
00265             this->ndcg[i] = 0.0;
00266         }
00267     }
00268 
00270     if (this->true_positives > 0) {
00271         this->average_precision = this->average_precision / (double)num_relevant_items;
00272         this->average_recall = this->average_recall / (double)num_relevant_items;
00273         this->average_dcg = this->average_dcg / (double)num_relevant_items;
00274         this->average_ndcg = this->average_ndcg / (double)num_relevant_items;
00275     } else {
00276         this->average_precision = 0.0;
00277         this->average_recall = 0.0;
00278         this->average_dcg = 0.0;
00279         this->average_ndcg = 0.0;
00280     }
00281 
00282     delete [] ideal_dcg;
00283 
00284 /*
00285 char str[100];
00286 //  sprintf(str, "%s - Topic %d", lst->get_voter()->get_name(), tid);
00287 
00288 if (RESULTS_TYPE == 1) {
00289 printf("| %s (%d/%d)\t| %5.4f | %5.4f | %5.4f | %5.4f | %5.4f | %5.4f | %5.4f | %5.4f | %5.4f
      | %5.4f  | %5.4f  |\n",
00290 PadStr(str, 10, ' '), this->true_positives, num_relevant_items, this->average_precision,
00291 num_items > 4 ?  this->precision[4] :  0,
```

```
00292 num_items > 9 ?  this->precision[9] :  0,
00293 num_items > 14 ?  this->precision[14] :  0,
00294 num_items > 19 ?  this->precision[19] :  0,
00295 num_items > 29 ?  this->precision[29] :  0,
00296 num_items > 99 ?  this->precision[99] :  0,
00297 num_items > 199 ?  this->precision[199] :  0,
00298 num_items > 499 ?  this->precision[499] :  0,
00299 num_items > 999 ?  this->precision[999] :  0,
00300 num_items > 9 ?  this->ndcg[9] :  0,
00301 num_items > 19 ?  this->ndcg[19] :  0);
00302
00303 } else if (RESULTS_TYPE == 2) {
00304 printf("| %s (%d/%d)\t| %4.3f | %4.3f | %4.3f | %4.3f | %4.3f | %4.3f | %4.3f | %4.3f | %4.3f
      | %4.3f | %4.3f | %4.3f | %4.3f |\n",
00305 PadStr(str, 10, ' '), this->true_positives, num_relevant_items, this->average_precision,
00306 num_items > 4 ?  this->precision[4] :  0,
00307 num_items > 9 ?  this->precision[9] :  0,
00308 num_items > 19 ?  this->precision[19] :  0,
00309 num_items > 29 ?  this->precision[29] :  0,
00310 num_items > 49 ?  this->precision[49] :  0,
00311 num_items > 99 ?  this->precision[99] :  0,
00312 this->average_ndcg,
00313 num_items > 4 ?  this->ndcg[4] :  0,
00314 num_items > 9 ?  this->ndcg[9] :  0,
00315 num_items > 19 ?  this->ndcg[19] :  0,
00316 num_items > 29 ?  this->ndcg[29] :  0,
00317 num_items > 49 ?  this->ndcg[49] :  0,
00318 num_items > 99 ?  this->ndcg[99] :  0);
00319 }
00320 */
00321     delete [] this->precision;
00322     this->precision = NULL;
00323     delete [] this->recall;
00324     this->recall = NULL;
00325     delete [] this->dcg;
00326     this->dcg = NULL;
00327     delete [] this->ndcg;
00328     this->ndcg = NULL;
00329
00330     return this->average_ndcg;
00331 }
00332
00333 void Evaluator::display_relevs() {
00334     this->relevs->display();
00335 }
00336
00338 uint32_t Evaluator::get_num_rel() { return this->relevs->get_num_nodes(); }
00339 uint32_t Evaluator::get_true_positives() { return this->true_positives; }
00340
00341 double Evaluator::get_average_precision() { return this->average_precision; }
00342 double Evaluator::get_average_recall() { return this->average_recall; }
00343 double Evaluator::get_average_dcg() { return this->average_dcg; }
00344 double Evaluator::get_average_ndcg() { return this->average_ndcg; }
00345
00346 double Evaluator::get_precision(uint32_t i) {
00347     if (!this->precision[i]) {
00348         return 0.0;
00349     }
00350     return this->precision[i];
00351 }
00352
00353 double Evaluator::get_recall(uint32_t i) {
00354     if (!this->recall[i]) {
00355         return 0.0;
00356     }
00357     return this->recall[i];
00358 }
00359
00360 double Evaluator::get_F1(uint32_t i) {
00361     if (!this->recall[i] || !this->precision[i]) {
00362         return 0.0;
00363     }
00364     return 2 * this->precision[i] * this->recall[i] / (this->precision[i] + this->recall[i]);
00365 }
00366
00367 double Evaluator::get_dcg(uint32_t i) {
00368     if (!this->dcg[i]) {
00369         return 0.0;
00370     }
00371     return this->dcg[i];
00372 }
00373
00374 inline double Evaluator::get_ndcg(uint32_t i) {
00375     if (!this->ndcg[i]) {
00376         return 0.0;
00377     }
00378     return this->ndcg[i];
```

```
00379 }
```

6.16 FLAGR/src/Evaluator.h File Reference

Classes

- class Evaluator

6.17 Evaluator.h

Go to the documentation of this file.

```
00001 #ifndef EVALUATOR_H
00002 #define EVALUATOR_H
00003
00004 class Evaluator {
00005     private:
00006         class Rels * relevs;
00007
00008         uint32_t true_positives;
00009
00010         double average_precision;
00011         double average_recall;
00012         double average_dcg;
00013         double average_ndcg;
00014
00015         double * precision;
00016         double * recall;
00017         double * dcg;
00018         double * ndcg;
00019
00020     private:
00021         static int cmp_dbl(const void *a, const void *b) {
00022             double x = *(double *)a;
00023             double y = *(double *)b;
00024             if (y >= x) {
00025                 return 1;
00026             }
00027             return -1;
00028         }
00029
00030     public:
00031         Evaluator();
00032         ~Evaluator();
00033
00034         void insert_relev(char *, uint32_t);
00035         void clear();
00036         void evaluate(rank_t, char *, class MergedList *, FILE *);
00037         double evaluate_input(class InputList *);
00038
00039         void display_relevs();
00040
00042         uint32_t get_num_rel();
00043         uint32_t get_true_positives();
00044
00045         double get_precision(uint32_t);
00046         double get_recall(uint32_t);
00047         double get_F1(uint32_t);
00048         double get_dcg(uint32_t);
00049         double get_ndcg(uint32_t);
00050
00051         double get_average_precision();
00052         double get_average_recall();
00053         double get_average_dcg();
00054         double get_average_ndcg();
00055 };
00056
00057 #endif // EVALUATOR_H
```

6.18 FLAGR/src/input/InputData.cpp File Reference

```
#include "InputData.h"
#include "InputDataCSV.cpp"
```

6.19 InputData.cpp

Go to the documentation of this file.

```
00001 #include "InputData.h"
00002
00003 // #include "InputDataTSV.cpp"
00004 #include "InputDataCSV.cpp"
00005
00007 InputData::InputData() :
00008         params(NULL),
00009         num_queries(0),
00010         queries(NULL),
00011         num_ret(0),
00012         num_rel(0),
00013         num_rel_ret(0),
00014         MAP(0.0),
00015         MNDCG(0.0),
00016         avg_sprho(0.0),
00017         mean_precision(NULL),
00018         mean_recall(NULL),
00019         mean_F1(NULL),
00020         mean_dcg(NULL),
00021         mean_ndcg(NULL),
00022         eval_file(NULL) { }
00023
00025 InputData::InputData(class InputParams * pr) :
00026         params(pr),
00027         num_queries(0),
00028         queries(NULL),
00029         num_ret(0),
00030         num_rel(0),
00031         num_rel_ret(0),
00032         MAP(0.0),
00033         MNDCG(0.0),
00034         avg_sprho(0.0),
00035         mean_precision(new double[MAX_LIST_ITEMS]),
00036         mean_recall(new double[MAX_LIST_ITEMS]),
00037         mean_F1(new double[MAX_LIST_ITEMS]),
00038         mean_dcg(new double[MAX_LIST_ITEMS]),
00039         mean_ndcg(new double[MAX_LIST_ITEMS]),
00040         eval_file(NULL) {
00041
00042             char * out;
00043             long file_size = 0;
00044
00045             if (pr->get_rels_file()) {
00046                 this->eval_file = fopen(pr->get_eval_file(), "w+");
00047                 this->initialize_stats();
00048             } else {
00049                 this->eval_file = NULL;
00050             }
00051
00053             FILE * datafile = fopen(this->params->get_input_file(), "r");
00054             if (datafile) {
00055                 out = this->read_file(datafile, &file_size);
00056
00057                 this->preprocess_CSV(out, file_size);
00058                 this->construct_CSV_lists(out, file_size);
00059
00060                 free(out);
00061                 fclose(datafile);
00062
00063                 for (uint32_t i = 0; i < this->num_queries; i++) {
00064                     this->queries[i]->init_weights();
00065                 }
00066
00067                 if (this->params->get_rels_file()) {
00068                     this->read_CSV_qrels();
00069                 }
00070             } else {
00071                 printf("Error Opening Input File %s\n", this->params->get_input_file());
00072                 exit(0);
00073             }
00074 }
00075
00077 InputData::~InputData() {
00078     if (this->num_queries > 0) {
00079         if (this->queries) {
00080             for (uint32_t i = 0; i < this->num_queries; i++) {
00081                 if (this->queries[i]) {
00082                     delete this->queries[i];
00083                 }
00084             }
00085             delete [] this->queries;
00086             if (this->mean_precision) { delete [] this->mean_precision; }
```

```
00087             if (this->mean_recall) { delete [] this->mean_recall; }
00088             if (this->mean_F1) { delete [] this->mean_F1; }
00089             if (this->mean_dcg) { delete [] this->mean_dcg; }
00090             if (this->mean_ndcg) { delete [] this->mean_ndcg; }
00091         }
00092     }
00093
00094     if (this->eval_file) {
00095         fclose(this->eval_file);
00096     }
00097 }
00098
00100 void InputData::initialize_stats() {
00101     this->avg_sprho = 0.0;
00102     this->MAP = 0.0;
00103     this->MNDCG = 0.0;
00104
00105     for (rank_t i = 0; i < MAX_LIST_ITEMS; i++) {
00106         this->mean_precision[i] = 0.0;
00107         this->mean_recall[i] = 0.0;
00108         this->mean_F1[i] = 0.0;
00109         this->mean_dcg[i] = 0.0;
00110         this->mean_ndcg[i] = 0.0;
00111     }
00112 }
00113
00115 void InputData::destroy_output_lists() {
00116     if (this->num_queries > 0) {
00117         if (this->queries) {
00118             for (uint32_t i = 0; i < this->num_queries; i++) {
00119                 if (this->queries[i]) {
00120                     this->queries[i]->destroy_output_list();
00121                 }
00122             }
00123         }
00124     }
00125 }
00126
00128 char * InputData::read_file(FILE * source, long * file_size) {
00129     fseek(source , 0L, SEEK_END);
00130     *file_size = ftell(source);
00131     rewind(source);
00132
00133     char * out = (char *)malloc((*file_size + 1) * sizeof(char));
00134
00135     if (fread(out, *file_size, 1 , source) != 1) {
00136         fclose(source);
00137         free(out);
00138         fputs("entire read fails",stderr);
00139         exit(1);
00140     }
00141     out[*file_size - 1] = 0;
00142
00143 //  printf("contents:  %s", out); getchar();
00144
00145     return out;
00146 }
00147
00149 void InputData::print_header() {
00150     char m1[1000], m3[1000], m4[1024], m5[1000];
00151     uint32_t ram = this->params->get_aggregation_method();
00152
00153     if (ram == 100) { strcpy(m1, "CombSUM with Borda normalization"); } else
00154     if (ram == 101) { strcpy(m1, "CombSUM with Rank normalization"); } else
00155     if (ram == 102) { strcpy(m1, "CombSUM with Score normalization"); } else
00156     if (ram == 103) { strcpy(m1, "CombSUM with Z-Score normalization"); } else
00157     if (ram == 104) { strcpy(m1, "CombSUM with simple Borda normalization"); } else
00158     if (ram == 110) { strcpy(m1, "CombMNZ with Borda normalization"); } else
00159     if (ram == 111) { strcpy(m1, "CombMNZ with Rank normalization"); } else
00160     if (ram == 112) { strcpy(m1, "CombMNZ with Score normalization"); } else
00161     if (ram == 113) { strcpy(m1, "CombMNZ with Z-Score normalization"); } else
00162     if (ram == 114) { strcpy(m1, "CombMNZ with simple Borda normalization"); } else
00163     if (ram == 200) { strcpy(m1, "Condorcet Winners Method"); } else
00164     if (ram == 201) { strcpy(m1, "Copeland Winners Method"); } else
00165     if (ram == 300) { strcpy(m1, "Outranking Approach"); } else
00166     if (ram == 400) { strcpy(m1, "Kemeny Optimal Aggregation"); } else
00167     if (ram == 401) { strcpy(m1, "Robust Rank Aggregation (RRA)"); } else
00168     if (ram == 5100) { strcpy(m1, "DIBRA @ CombSUM with Borda normalization"); } else
00169     if (ram == 5101) { strcpy(m1, "DIBRA @ CombSUM with Rank normalization"); } else
00170     if (ram == 5102) { strcpy(m1, "DIBRA @ CombSUM with Score normalization"); } else
00171     if (ram == 5103) { strcpy(m1, "DIBRA @ CombSUM with Z-Score normalization"); } else
00172     if (ram == 5104) { strcpy(m1, "DIBRA @ CombSUM with simple Borda normalization"); } else
00173     if (ram == 5110) { strcpy(m1, "DIBRA @ CombMNZ with Borda normalization"); } else
00174     if (ram == 5111) { strcpy(m1, "DIBRA @ CombMNZ with Rank normalization"); } else
00175     if (ram == 5112) { strcpy(m1, "DIBRA @ CombMNZ with Score normalization"); } else
00176     if (ram == 5113) { strcpy(m1, "DIBRA @ CombMNZ with Z-Score normalization"); } else
00177     if (ram == 5114) { strcpy(m1, "DIBRA @ CombMNZ with simple Borda normalization"); } else
```

```
00178     if (ram == 5200) { strcpy(m1, "DIBRA @ Condorcet Winners Method"); } else
00179     if (ram == 5201) { strcpy(m1, "DIBRA @ Copeland Winners Method"); } else
00180     if (ram == 5300) { strcpy(m1, "DIBRA @ Outranking Approach"); } else
00181     if (ram == 600) { strcpy(m1, "Preference Relations Method"); } else
00182     if (ram == 700) { strcpy(m1, "Agglomerative Aggregation"); } else
00183     if (ram == 801) { strcpy(m1, "Markov Chains 1"); } else
00184     if (ram == 802) { strcpy(m1, "Markov Chains 2"); } else
00185     if (ram == 803) { strcpy(m1, "Markov Chains 3"); } else
00186     if (ram == 804) { strcpy(m1, "Markov Chains 4"); } else
00187     if (ram == 805) { strcpy(m1, "MCT"); }
00188
00189     printf("| %s / %s", m1, this->params->get_input_file());
00190
00191     if (this->params->get_aggregation_method() == 5 || this->params->get_aggregation_method() == 6 ||
00192         this->params->get_aggregation_method() == 7 || this->params->get_aggregation_method() == 8) {
00193
00194         if (params->get_correlation_method() == 1) { strcpy(m3, "Spearman's Rho correlation"); } else
00195         if (params->get_correlation_method() == 2) { strcpy(m3, "Scaled Footrule Distance"); } else
00196         if (params->get_correlation_method() == 3) { strcpy(m3, "Weighted Cosine Distance"); } else
00197         if (params->get_correlation_method() == 4) { strcpy(m3, "Local Scaled Footrule Distance"); }
      else
00198         if (params->get_correlation_method() == 5) { strcpy(m3, "Kendall's Tau correlation"); }
00199
00200         sprintf(m4, " - %s  - g = %3.1f)", m3, this->params->get_gamma());
00201
00202     } else if (this->params->get_aggregation_method() == 9) {
00203         sprintf(m4, " - a = %3.1f, b = %3.1f)", this->params->get_alpha(), this->params->get_beta());
00204     } else {
00205         strcpy(m4, ")");
00206     }
00207
00208     if (params->get_list_pruning() == 1) {
00209         strcpy(m5, "List Pruning Enabled.");
00210     } else {
00211         strcpy(m5, "List Pruning Disabled.");
00212     }
00213     printf("%s - %s\n", m4, m5); fflush(NULL);
00214
00215     if (this->eval_file) {
00216         fprintf(this->eval_file, "| %s / %s", m1, this->params->get_input_file());
00217         fprintf(this->eval_file, "%s - %s\n", m4, m5); fflush(NULL);
00218     }
00219 }
00220
00221
00224 void InputData::evaluate() {
00225     rank_t max_pts = this->params->get_eval_points();
00226     double precision_acc[max_pts], recall_acc[max_pts], F1_acc[max_pts], dcg_acc[max_pts],
      ndcg_acc[max_pts];
00227     double sum_avep = 0.0, sum_aver = 0.0, sum_aved = 0.0, sum_aven = 0.0;
00228     uint32_t m = 0;
00229     rank_t i = 0, cutoff = 0;
00230
00232     this->initialize_stats();
00233
00234     for (i = 0; i < max_pts; i++) {
00235         precision_acc[i] = 0.0;
00236         recall_acc[i] = 0.0;
00237         F1_acc[i] = 0.0;
00238         dcg_acc[i] = 0.0;
00239         ndcg_acc[i] = 0.0;
00240     }
00241
00243     fprintf(eval_file, "q,num_ret,num_rel,num_rel_ret,map,");
00244     for (i = 0; i < max_pts; i++) { fprintf(eval_file, "P_%d,", i + 1); }
00245     for (i = 0; i < max_pts; i++) { fprintf(eval_file, "recall_%d,", i + 1); }
00246     for (i = 0; i < max_pts; i++) { fprintf(eval_file, "dcg_cut_%d,", i + 1); }
00247     for (i = 0; i < max_pts; i++) {
00248         if (i < max_pts - 1) {
00249             fprintf(eval_file, "ndcg_cut_%d,", i + 1);
00250         } else {
00251             fprintf(eval_file, "ndcg_cut_%d", i + 1);
00252         }
00253     }
00254     fprintf(eval_file, "\n");
00255
00257     for (m = 0; m < this->num_queries; m++) {
00258 //      printf("Evaluating Query %d:  %s (%d items)\n", m + 1,
00259 //              this->queries[m]->get_topic(), this->queries[m]->get_num_items());
00260
00261         this->queries[m]->evaluate(max_pts, this->eval_file);
00262
00263         if (this->queries[m]->get_num_items() > max_pts) {
00264             cutoff = max_pts;
00265         } else {
00266             cutoff = this->queries[m]->get_num_items();
00267         }
```

```
00268
00269         this->num_ret += this->queries[m]->get_num_items();
00270         this->num_rel += this->queries[m]->get_num_rel();
00271         this->num_rel_ret += this->queries[m]->get_num_rel_ret();
00272
00274         for (i = 0; i < cutoff; i++) {
00275             precision_acc[i] += this->queries[m]->get_precision(i);
00276             recall_acc[i] += this->queries[m]->get_recall(i);
00277             F1_acc[i] += this->queries[m]->get_F1(i);
00278             dcg_acc[i] += this->queries[m]->get_dcg(i);
00279             ndcg_acc[i] += this->queries[m]->get_ndcg(i);
00280         }
00281
00282         sum_avep += this->queries[m]->get_average_precision();
00283         sum_aver += this->queries[m]->get_average_recall();
00284         sum_aved += this->queries[m]->get_average_dcg();
00285         sum_aven += this->queries[m]->get_average_ndcg();
00286
00287         this->avg_sprho += this->queries[m]->evaluate_experts_list();
00288     }
00289
00291     for (i = 0; i < max_pts; i++) {
00292         this->mean_precision[i] = precision_acc[i] / (double) this->num_queries;
00293         this->mean_recall[i] = recall_acc[i] / (double) this->num_queries;
00294         this->mean_F1[i] = F1_acc[i] / (double) this->num_queries;
00295         this->mean_dcg[i] = dcg_acc[i] / (double) this->num_queries;
00296         this->mean_ndcg[i] = ndcg_acc[i] / (double) this->num_queries;
00297     }
00298     this->MAP = sum_avep / (double) this->num_queries;
00299     this->MNDCG = sum_aven / (double) this->num_queries;
00300     this->avg_sprho = this->avg_sprho / (double) this->num_queries;
00301
00303     fprintf(eval_file, "all,%d,%d,%d,%7.6f,",
00304             this->num_ret, this->num_rel, this->num_rel_ret, this->MAP);
00305
00306     for (i = 0; i < max_pts; i++) { fprintf(eval_file, "%7.6f,", this->mean_precision[i]); }
00307     for (i = 0; i < max_pts; i++) { fprintf(eval_file, "%7.6f,", this->mean_recall[i]); }
00308     for (i = 0; i < max_pts; i++) { fprintf(eval_file, "%7.6f,", this->mean_dcg[i]); }
00309     for (i = 0; i < max_pts; i++) {
00310         if (i < max_pts - 1) {
00311             fprintf(eval_file, "%7.6f,", this->mean_ndcg[i]);
00312         } else {
00313             fprintf(eval_file, "%7.6f", this->mean_ndcg[i]);
00314         }
00315     }
00316     fprintf(eval_file, "\n");
00317 }
00318
00319
00321 void InputData::evaluate_input() {
00322     uint32_t m = 0;
00323
00325     this->initialize_stats();
00326
00327     for (m = 0; m < this->num_queries; m++) {
00328         this->queries[m]->evaluate_input();
00329     }
00330 }
00331
00333 void InputData::aggregate() {
00334 /*
00335 printf("\nPerforming Aggregation...\nParameters ");
00336 this->print_header();
00337 printf("\n");
00338 */
00339     for (uint32_t m = 0; m < this->num_queries; m++) {
00340 //      printf("Processing Query %d / %d...  [ ", m + 1, this->num_queries); fflush(NULL);
00341         this->queries[m]->aggregate(this->params);
00342 //      printf(" OK ]\n"); fflush(NULL);
00343     }
00344 }
00345
00347 uint32_t InputData::compute_avg_list_length() {
00348     uint32_t sum = 0, sum2 = 0;
00349     for (uint32_t q = 0; q < this->num_queries; q++) {
00350         for (uint32_t l = 0; l < this->queries[q]->get_num_input_lists(); l++) {
00351             sum2++;
00352             sum += this->queries[q]->get_input_list(l)->get_num_items();
00353         }
00354     }
00355     return sum / sum2;
00356 }
00357
00359 uint32_t InputData::get_num_queries() { return this->num_queries; }
00360 class Query * InputData::get_query(uint32_t i) { return this->queries[i]; }
00361 double InputData::get_mean_precision(uint32_t i) { return this->mean_precision[i]; }
00362 double InputData::get_mean_recall(uint32_t i) { return this->mean_recall[i]; }
```

```
00363 double InputData::get_mean_F1(uint32_t i) { return this->mean_F1[i]; }
00364 double InputData::get_mean_dcg(uint32_t i) { return this->mean_dcg[i]; }
00365 double InputData::get_mean_ndcg(uint32_t i) { return this->mean_ndcg[i]; }
00366 double InputData::get_MAP() { return this->MAP; }
00367 double InputData::get_MNDCG() { return this->MNDCG; }
00368 double InputData::get_avg_sprho() { return this->avg_sprho; }
00369 rank_t InputData::get_num_ret() { return this->num_ret; }
00370 rank_t InputData::get_num_rel() { return this->num_rel; }
00371 rank_t InputData::get_num_rel_ret() { return this->num_rel_ret; }
```

6.20 FLAGR/src/input/InputData.h File Reference

Classes

- class InputData

6.21 InputData.h

Go to the documentation of this file.

```
00001 #ifndef INPUTDATA_H
00002 #define INPUTDATA_H
00003
00004 class InputData {
00005     private:
00006         class InputParams * params;
00007
00008         uint32_t num_queries;
00009         class Query ** queries;
00010
00012         rank_t num_ret;
00013         rank_t num_rel;
00014         rank_t num_rel_ret;
00015
00016         double MAP;
00017         double MNDCG;
00018         double avg_sprho;
00019         double * mean_precision;
00020         double * mean_recall;
00021         double * mean_F1;
00022         double * mean_dcg;
00023         double * mean_ndcg;
00024
00025         FILE * eval_file;
00026
00027     private:
00028         char * read_file(FILE *, long *);
00029         void get_TSV_queries(char *, uint32_t);
00030         void process_TSV_lists(char *, uint32_t, char *);
00031         void read_TSV_qrels();
00032         void read_CSV_qrels();
00033
00034         void preprocess_CSV(char *, uint32_t);
00035         void construct_CSV_lists(char *, uint32_t);
00036
00037         void initialize_stats();
00038
00039     public:
00040         InputData();
00041         InputData(class InputParams * PARAMS);
00042         ~InputData();
00043
00044         void aggregate();
00045         void evaluate();
00046         void evaluate_input();
00047         void destroy_output_lists();
00048
00049         void print_header();
00050
00052         uint32_t get_num_queries();
00053         class Query * get_query(uint32_t);
00054
00055         double get_MAP();
00056         double get_MNDCG();
00057         rank_t get_num_ret();
```

```
00058         rank_t get_num_rel();
00059         rank_t get_num_rel_ret();
00060         double get_mean_precision(uint32_t);
00061         double get_mean_recall(uint32_t);
00062         double get_mean_F1(uint32_t);
00063         double get_mean_dcg(uint32_t);
00064         double get_mean_ndcg(uint32_t);
00065         double get_avg_sprho();
00066         FILE * get_eval_file();
00067         uint32_t compute_avg_list_length();
00068 };
00069
00070 #endif // TRECINPUT_H
```

6.22 FLAGR/src/input/InputDataCSV.cpp File Reference

6.23 InputDataCSV.cpp

Go to the documentation of this file.

```
00001
00002 void InputData::preprocess_CSV(char * in, uint32_t len) {
00003     uint32_t i = 0, occ = 0, y = 0, q = 0, v = 0;
00004     uint32_t num_alloc_topics = 10, num_alloc_voters = 4, num_voters = 0;
00005
00006     char buf[100], prev_topic[100], prev_voter[100], topic[100], voter[100];
00007     bool found;
00008
00009     buf[0] = 0;
00010     topic[0] = 0;
00011     voter[0] = 0;
00012     prev_topic[0] = 0;
00013     prev_voter[0] = 0;
00014
00016     char ** query_strings = (char**)malloc(num_alloc_topics * sizeof(char *));
00017     char ** voter_strings = (char**)malloc(num_alloc_voters * sizeof(char *));
00018
00020     for (i = 0; i < len; i++) {
00021
00023         if (in[i] == 44) {
00024
00026             if (occ == 0) {
00027                 buf[y] = 0;
00028                 strcpy(topic, buf);
00029
00030                 if (strcmp(topic, prev_topic) != 0) {
00031                     strcpy(prev_topic, topic);
00032 //                  printf("Topic:  %s\n", buf); fflush(NULL); getchar();
00033                     found = false;
00034                     for (q = 0; q < this->num_queries; q++) {
00035                         if (strcmp(buf, query_strings[q]) == 0) {
00036                             found = true;
00037                             break;
00038                         }
00039                     }
00040
00041                     if (!found) {
00042                         query_strings[this->num_queries] = (char *) malloc((y + 1) * sizeof(char));
00043                         strcpy(query_strings[this->num_queries], buf);
00044                         this->num_queries++;
00045
00046                         if (this->num_queries >= num_alloc_topics) {
00047                             num_alloc_topics *= 2;
00048                             query_strings = (char **)realloc
00049                                 (query_strings, num_alloc_topics * sizeof(char *));
00050                         }
00051                     }
00052                 }
00053
00054                 occ = 1;
00055                 y = 0;
00056             }
00057
00059             else if (occ == 1) {
00060                 buf[y] = 0;
00061                 strcpy(voter, buf);
00062
00063                 if (strcmp(voter, prev_voter) != 0) {
00064                     strcpy(prev_voter, voter);
```

```
00065                     found = false;
00066                     for (v = 0; v < num_voters; v++) {
00067                         if (strcmp(buf, voter_strings[v]) == 0) {
00068                             found = true;
00069                             break;
00070                         }
00071                     }
00072 
00073                     if (!found) {
00074                         voter_strings[num_voters] = (char *) malloc((y + 1) * sizeof(char));
00075                         strcpy(voter_strings[num_voters], buf);
00076                         num_voters++;
00077 
00078                         if (num_voters >= num_alloc_voters) {
00079                             num_alloc_voters *= 2;
00080                             voter_strings = (char **)realloc
00081                                 (voter_strings, num_alloc_voters * sizeof(char *));
00082                         }
00083                     }
00084                 }
00085 
00086                 occ = 2;
00087                 y = 0;
00088 
00090             } else {
00091                 y = 0;
00092             }
00093 
00095         } else if (in[i] == 10) {
00096             occ = 0;
00097             y = 0;
00098 
00100         } else {
00101             buf[y++] = in[i];
00102         }
00103     }
00104 
00107     this->queries = new Query * [this->num_queries];
00108     for (q = 0; q < this->num_queries; q++) {
00109         this->queries[q] = new Query(1);
00110 //      this->queries[q]->set_topic_id(atoi(query_strings[q]));
00111         this->queries[q]->set_topic(query_strings[q]);
00112 
00113         for (v = 0; v < num_voters; v++) {
00114             this->queries[q]->create_list(voter_strings[v], 1.00);
00115         }
00116     }
00117 
00119     for (q = 0; q < this->num_queries; q++) { free(query_strings[q]); }
00120     for (v = 0; v < num_voters; v++) { free(voter_strings[v]); }
00121     free(query_strings);
00122     free(voter_strings);
00123 }
00124 
00127 void InputData::construct_CSV_lists(char * in, uint32_t len) {
00128     uint32_t i = 0, occ = 0, y = 0, q = 0, v = 0, active_query = 0, active_voter = 0;
00129     score_t scr;
00130     char buf[100], prev_topic[100], prev_voter[100], topic[100], voter[100], code_c[100];
00131     class InputList * inlist = NULL;
00132 
00133     buf[0] = 0;
00134     topic[0] = 0;
00135     voter[0] = 0;
00136     prev_topic[0] = 0;
00137     prev_voter[0] = 0;
00138 
00140     for (i = 0; i < len; i++) {
00141 
00143         if (in[i] == 44) {
00144 
00146             if (occ == 0) {
00147                 buf[y] = 0;
00148                 strcpy(topic, buf);
00149 
00150                 if (strcmp(topic, prev_topic) != 0) {
00151                     strcpy(prev_topic, topic);
00152 //                  printf("Topic:  %s\n", buf); fflush(NULL); getchar();
00153                     for (q = 0; q < this->num_queries; q++) {
00154                         if (strcmp(topic, this->queries[q]->get_topic()) == 0) {
00155                             active_query = q;
00156                             break;
00157                         }
00158                     }
00159                 }
00160 
00161                 occ = 1;
00162                 y = 0;
```

```
00163             }
00164
00166             else if (occ == 1) {
00167                 buf[y] = 0;
00168                 strcpy(voter, buf);
00169
00173                 if (strcmp(voter, prev_voter) != 0) {
00174                     strcpy(prev_voter, voter);
00175                     for (v = 0; v < this->queries[active_query]->get_num_input_lists(); v++) {
00176                         if (strcmp(buf,
      this->queries[active_query]->get_input_list(v)->get_voter()->get_name()) == 0) {
00177                             active_voter = v;
00178                             break;
00179                         }
00180                     }
00181                 }
00182
00183                 inlist = this->queries[active_query]->get_input_list(active_voter);
00184                 occ = 2;
00185                 y = 0;
00186
00188             } else if (occ == 2) {
00189                 buf[y] = 0;
00190                 strcpy(code_c, buf);
00191                 occ = 3;
00192                 y = 0;
00193
00195             } else if (occ == 3) {
00196                 buf[y] = 0;
00197                 scr = strtod(buf, NULL);
00198
00199 //              printf("Inserting %s with score:  %5.3f\n", code_c, scr); getchar();
00200                 inlist->insert_item(code_c, 0, scr);
00201
00202                 occ = 4;
00203                 y = 0;
00204
00205             } else {
00206                 y = 0;
00207             }
00208
00210         } else if (in[i] == 10) {
00211             occ = 0;
00212             y = 0;
00213
00215         } else {
00216             if (in[i] != 10) {
00217                 buf[y++] = in[i];
00218             }
00219         }
00220     }
00221
00223     for (q = 0; q < this->num_queries; q++) {
00224         for (v = 0; v < this->queries[q]->get_num_input_lists(); v++) {
00225             this->queries[q]->get_input_list(v)->sort_by_score();
00226 //          this->queries[q]->get_input_list(v)->display();
00227         }
00228     }
00229 }
00230
00232 void InputData::read_CSV_qrels() {
00233     uint32_t i = 0, occ = 0, y = 0, q = 0, active_query = 0, rel_judg = 0;
00234     size_t nread = 0, file_size = 0;
00235
00236     char buf[100], prev_topic[100], topic[100], code_c[100], rj_c[100];
00237     char * in = NULL;
00238
00239     buf[0] = 0;
00240     topic[0] = 0;
00241     prev_topic[0] = 0;
00242     code_c[0] = 0;
00243     rj_c[0] = 0;
00244
00245     FILE * fp = fopen(this->params->get_rels_file(), "r");
00246     if (fp) {
00247         fseek(fp, 0L, SEEK_END);
00248         file_size = ftell(fp);
00249         rewind(fp);
00250
00251         in = (char *)malloc( (file_size + 2) * sizeof(char));
00252         nread = fread(in, sizeof(char), file_size, fp);
00253         in[nread] = 0;
00254
00255
00256         for (i = 0; i < file_size; i++) {
00257
00259             if (in[i] == 44) {
```

```
00260
00262                 if (occ == 0) {
00263                     buf[y] = 0;
00264                     strcpy(topic, buf);
00265                     occ = 1;
00266                     y = 0;
00267                 }
00268
00269
00271                 else if (occ == 1) {
00272                     buf[y] = 0;
00273                     occ = 2;
00274                     y = 0;
00275                 }
00276
00278                 else if (occ == 2) {
00279                     buf[y] = 0;
00280                     strcpy(code_c, buf);
00281                     occ = 3;
00282                     y = 0;
00283                 }
00284
00286             } else if (in[i] == 10) {
00287                 buf[y] = 0;
00288                 strcpy(rj_c, buf);
00289                 rel_judg = atoi(rj_c);
00290
00291                 occ = 0;
00292                 y = 0;
00293
00294                 if (strcmp(topic, prev_topic) != 0) {
00295                     strcpy(prev_topic, topic);
00296
00297                     for (q = 0; q < this->num_queries; q++) {
00298                         if (strcmp(topic, this->queries[q]->get_topic()) == 0) {
00299                             active_query = q;
00300                             break;
00301                         }
00302                     }
00303                 }
00304
00305                 if (rel_judg > 0) {
00306 //                  printf("\tQuery %d (%s) => Inserting:  %s - relevance:  d\n",
00307 //                                  active_query, topic, code_c, rel_judg);
00308                     this->queries[active_query]->insert_relev(code_c, rel_judg);
00309                 }
00310
00312             } else {
00313                 buf[y++] = in[i];
00314             }
00315         }
00316
00317         free(in);
00318         fclose(fp);
00319     } else {
00320         printf("Error opening rels file %s.  Continuing without evaluation\n",
00321                 this->params->get_rels_file());
00322         fflush(NULL);
00323     }
00324 }
```

6.24 FLAGR/src/input/InputDataTSV.cpp File Reference

6.25 InputDataTSV.cpp

Go to the documentation of this file.

```
00001
00002 void InputData::get_TSV_queries(char * in, uint32_t len) {
00003     uint32_t i = 0, occ = 0, y = 0;
00004     int32_t topic = 0, prev_topic = 0;
00005     char buf[100];
00006
00007     for (i = 0; i < len; i++) {
00008         if (in[i] == 32) {
00009             while (in[i] != 32) { i++; }
00010         } else if (in[i] == 10) {
00011             occ = 0;
00012
```

```
00013             if (topic != prev_topic) {
00014 //                printf("New topic %d\n", topic);
00015                 this->num_queries++;
00016                 prev_topic = topic;
00017             }
00018
00019         } else {
00020
00022             if (occ == 0) {
00023                 y = 0;
00024                 while (in[i] != 9 && in[i] != 32) { buf[y++] = in[i++]; }
00025                 buf[y] = 0;
00026                 occ = 1;
00027                 topic = atoi(buf);
00028                 i++;
00029             }
00030
00032             if (occ == 1) {
00033                 y = 0;
00034                 while (in[i] != 9 && in[i] != 32) { i++; y++; }
00035                 occ = 2;
00036                 i++;
00037             }
00038
00040             if (occ == 2) {
00041                 y = 0;
00042                 while (in[i] != 9 && in[i] != 32) { i++; y++; }
00043                 occ = 3;
00044                 i++;
00045             }
00046
00048             if (occ == 3) {
00049                 y = 0;
00050                 while (in[i] != 9 && in[i] != 32) { y++; i++; }
00051                 occ = 4;
00052                 i++;
00053                 if (prev_topic == 0) {
00054                     this->num_queries++;
00055                     prev_topic = topic;
00056                 }
00057             }
00058         }
00059     }
00060 }
00061
00062
00064 void InputData::process_TSV_lists(char * in, uint32_t len, char * voter) {
00065     uint32_t i = 0, occ = 0, y = 0, line = 0, qctr = 0;
00066     rank_t rank = 0;
00067     char topic_id[100], code_c[100], rank_c[100];
00068     int32_t topic = 0, prev_topic = 0;
00069     bool add_1 = false;
00070
00071     class InputList * inlist = NULL;
00072
00073     for (i = 0; i < len; i++) {
00074         if(in[i] == 32) {
00075             while (in[i] != 32) {
00076                 i++;
00077             }
00078         } else if (in[i] == 10) {
00079             occ = 0;
00080
00081             line++;
00082             if (topic != prev_topic) {
00083 //                printf("New topic %d - Rank:  %d - List Items:  %d\n", topic, rank,
      inlist->get_num_items());
00084                 inlist->sort_by_ranking();
00085 //                inlist->display();
00086
00087                 if (qctr >= this->num_queries) {
00088                     printf("Error!  Cannot accommodate more than %d queries\n", qctr);
00089                     break;
00090                 }
00091
00092                 this->queries[qctr]->set_topic_id(topic);
00093
00094                 inlist = this->queries[qctr++]->create_list(voter, 1.00);
00095
00096                 prev_topic = topic;
00097
00098                 if (rank == 0) { add_1 = true; } else { add_1 = false; }
00099 //                rank = 1;
00100
00101 //                getchar();
00102             }
00103
```

```
00104             if (strcmp(code_c, "") != 0) {
00105                 if (add_1) { rank++; }
00106 
00107 //                printf("Topic:  %s (%d).  Rank=%d.  URL=%s, Line:  %d\n", topic_id, qctr, rank,
      code_c, line);
00108 
00109                 inlist->insert_item(code_c, rank);
00110             }
00111 
00112         } else {
00113 
00115             if (occ == 0) {
00116                 y = 0;
00117                 while (in[i] != 9 && in[i] != 32) { topic_id[y++] = in[i++]; }
00118                 topic_id[y] = 0;
00119                 occ = 1;
00120                 topic = atoi(topic_id);
00121                 i++;
00122             }
00123 
00125             if (occ == 1) {
00126                 y = 0;
00127                 while (in[i] != 9 && in[i] != 32) {
00128                     i++;
00129                     y++;
00130                 }
00131                 occ = 2;
00132                 i++;
00133             }
00134 
00136             if (occ == 2) {
00137                 y = 0;
00138                 while (in[i] != 9 && in[i] != 32) {
00139                     if (in[i] >= 65 && in[i] <= 90) {
00140                         code_c[y++] = in[i++] + 32;
00141                     } else {
00142                         code_c[y++] = in[i++];
00143                     }
00144                 }
00145                 code_c[y] = 0;
00146                 occ = 3;
00147                 i++;
00148             }
00149 
00151             if (occ == 3) {
00152                 y = 0;
00153                 while (in[i] != 9 && in[i] != 32) { rank_c[y++] = in[i++]; }
00154                 rank_c[y] = 0;
00155                 rank = atoi(rank_c);
00156                 occ = 4;
00157                 i++;
00158                 if (prev_topic == 0) {
00159                     if (rank == 0) { add_1 = true; }
00160                     this->queries[qctr]->set_topic_id(topic);
00161                     inlist = this->queries[qctr++]->create_list(voter, 1.00);
00162                     prev_topic = topic;
00163                 }
00164             }
00165         }
00166     }
00167     inlist->sort_by_ranking();
00168 }
00169 
00171 void InputData::read_TSV_qrels() {
00172     uint32_t file_size = 0, i = 0, y = 0, occ = 0, line = 0, qctr = 0;
00173     int32_t rj = 0;
00174     uint32_t topic = 0, prev_topic = 0;
00175 
00176     char qrels_filepath[1024], topic_id[100], rj_c[100], code_c[100], * buf, u_c[100];
00177     size_t nread = 0;
00178 
00179     sprintf(qrels_filepath, "%s%s/%s_%s_prels", this->params->get_base_path(),
00180         this->params->get_dataset_name(), this->params->get_dataset_name(),
      this->params->get_dataset_track());
00181 
00182     FILE * fp = fopen(qrels_filepath, "r");
00183     if (fp) {
00184         printf("Reading relevance judgments...\n"); fflush(NULL);
00185         fseek(fp, 0L, SEEK_END);
00186         file_size = ftell(fp);
00187         rewind(fp);
00188 
00189         buf = (char *)malloc( (file_size + 2) * sizeof(char));
00190         nread = fread(buf, sizeof(char), file_size, fp);
00191         buf[nread] = 0;
00192 
00193         for (i = 0; i < file_size; i++) {
```

```
00194             if(buf[i] == 32) {
00195                 while (buf[i] == 32) {
00196                     i++;
00197                 }
00198                 i--;
00199             } else if (buf[i] == 10) {
00200                 occ = 0;
00201 
00202                 line++;
00203                 if (topic != prev_topic) {
00204                     prev_topic = topic;
00205                     for (qctr = 0; qctr < this->num_queries; qctr++) {
00206                         if (this->queries[qctr]->get_topic_id() == topic) {
00207                             break;
00208                         }
00209                     }
00210                 }
00211 
00212 //                printf("Topic:  %s.  Line=%d.  URL=%s, Judge:  %s\n", topic_id, line, code_c, rj_c);
00213                 if (strcmp(code_c, "") != 0) {
00214                     if (rj > 0) {
00215                         this->queries[qctr]->insert_relev(topic, code_c, rj);
00216 //                        printf("Inserted %d (%s) into query %d (Topic:  %d - %d)\n",
00217 //                            this->queries[qctr]->get_num_relevs(), code_c, qctr,
00218 //                            this->queries[qctr]->get_topic_id(), topic);
00219                     }
00220                 }
00221             } else {
00222 
00224                 if (occ == 0) {
00225                     y = 0;
00226                     while (buf[i] != 9 && buf[i] != 32 && buf[i] != 10) { topic_id[y++] = buf[i++]; }
00227                     topic_id[y] = 0;
00228 
00230                     if (strcmp(this->params->get_dataset_name(), "TREC2009") == 0) {
00231                         occ = 2;
00232                     } else {
00233                         occ = 1;
00234                     }
00235 
00236                     topic = atoi(topic_id);
00237                     if (prev_topic == 0) {
00238                         prev_topic = topic;
00239                     }
00240                 }
00241 
00243                 else if (occ == 1) {
00244                     y = 0;
00245                     while (buf[i] != 9 && buf[i] != 32 && buf[i] != 10) { u_c[y++] = buf[i++]; }
00246                     u_c[y] = 0;
00247                     occ = 2;
00248                 }
00249 
00251                 else if (occ == 2) {
00252                     y = 0;
00253                     while (buf[i] != 9 && buf[i] != 32 && buf[i] != 10) {
00254                         if (buf[i] >= 65 && buf[i] <= 90) {
00255                             code_c[y++] = buf[i++] + 32;
00256                         } else {
00257                             code_c[y++] = buf[i++];
00258                         }
00259                     }
00260                     code_c[y] = 0;
00261                     occ = 3;
00262                 }
00263 
00265                 else if (occ == 3) {
00266                     y = 0;
00267                     while (buf[i] != 9 && buf[i] != 32 && buf[i] != 10) { rj_c[y++] = buf[i++]; }
00268 
00269                     rj_c[y] = 0;
00270                     rj = atoi(rj_c);
00271                     occ = 4;
00272                     i--;
00273                 }
00274             }
00275         }
00276         free(buf);
00277 
00278         fclose(fp);
00279     } else {
00280         printf("Error opening qrels file %s\n", qrels_filepath);
00281         exit(-1);
00282     }
00283 }
00284 
```

6.26 FLAGR/src/InputItem.cpp File Reference

```
#include "InputItem.h"
```

6.27 InputItem.cpp

Go to the documentation of this file.

```
00001 #include "InputItem.h"
00002
00004 InputItem::InputItem() :  code(NULL), rank(0), inscore(0.0) { }
00005
00007 InputItem::InputItem(char * c, rank_t r, score_t s) :  code(NULL), rank(r), inscore(s) {
00008     this->copy_code(c);
00009 }
00010
00012 InputItem::~InputItem() {
00013     if (this->code) {
00014         delete [] this->code;
00015     }
00016 }
00017
00019 void InputItem::copy_code(char *v) {
00020     this->code = new char[strlen(v) + 1];
00021     strcpy(this->code, v);
00022 }
00023
00025 void InputItem::display() {
00026     printf("ITEM: %s, RANKING: %d, SCORE: %5.3f\n", this->code, this->rank, this->inscore);
00027 }
00028
00030 void InputItem::set_code(char * v) { this->copy_code(v); }
00031 void InputItem::set_rank(rank_t v) { this->rank = v; }
00032 void InputItem::set_inscore(score_t v) { this->inscore = v; }
00033
00035 char * InputItem::get_code() { return this->code; }
00036 rank_t InputItem::get_rank() { return this->rank; }
00037 score_t InputItem::get_inscore() { return this->inscore; }
```

6.28 FLAGR/src/InputItem.h File Reference

Classes

- class InputItem

6.29 InputItem.h

Go to the documentation of this file.

```
00001 #ifndef INPUTITEM_H
00002 #define INPUTITEM_H
00003
00004
00005 class InputItem {
00006     protected:
00007         char * code;
00008         rank_t rank;
00009         score_t inscore;
00010
00011     private:
00012         void copy_code(char *);
00013
00014     public:
00015         InputItem();
00016         InputItem(char *, rank_t, score_t);
00017
```

```
00018         ~InputItem();
00019
00020         void display();
00021
00023         void set_code(char *);
00024         void set_rank(rank_t);
00025         void set_inscore(score_t);
00026
00028         char * get_code();
00029         rank_t get_rank();
00030         score_t get_inscore();
00031 };
00032
00033 #endif // INPUTITEM_H
```

6.30 FLAGR/src/InputList.cpp File Reference

```
#include "InputList.h"
```

6.31 InputList.cpp

Go to the documentation of this file.

```
00001 #include "InputList.h"
00002
00004 InputList::InputList() :
00005         id(0),
00006         voter(NULL),
00007         num_items(0),
00008         num_alloc_items(0),
00009         cutoff(0),
00010         stats(NULL),
00011         items(NULL) { }
00012
00014 InputList::InputList(uint32_t i, char * v, score_t w) :
00015         id(i),
00016         voter(new Voter(v, w)),
00017         num_items(0),
00018         num_alloc_items(100),
00019         cutoff(0),
00020         stats(new SimpleScoreStats()),
00021         items((class InputItem **)malloc(this->num_alloc_items * sizeof(class InputItem *))) { }
00022
00024 InputList::~InputList() {
00025     if (this->voter) {
00026         delete this->voter;
00027     }
00028
00029     if (this->items) {
00030         for (rank_t i = 0; i < this->num_items; i++) {
00031             if (this->items[i]) {
00032                 delete this->items[i];
00033             }
00034         }
00035         free(this->items);
00036     }
00037
00038     if (this->stats) {
00039         delete this->stats;
00040     }
00041 }
00042
00044 void InputList::insert_item(char * code, rank_t r, score_t s) {
00045     if (this->num_items <= MAX_LIST_ITEMS) {
00046         this->items[this->num_items] = new InputItem(code, r, s);
00047         this->num_items++;
00048         this->cutoff++;
00049
00050         if (this->num_items >= this->num_alloc_items) {
00051             this->num_alloc_items *= 2;
00052             this->items = (class InputItem **)realloc
00053                     (this->items, this->num_alloc_items * sizeof(class InputItem *));
00054         }
00055     }
00056 }
```

```
00057
00059 void InputList::replace_item(char * code, rank_t r, score_t s) {
00060     delete this->items[r - 1];
00061     this->items[r - 1] = new InputItem(code, r, s);
00062 }
00063
00065 class InputItem * InputList::search_item(char * code) {
00066     for (uint32_t i = 0; i < this->num_items; i++) {
00067         if ( strcmp(this->items[i]->get_code(), code) == 0 ) {
00068             return this->items[i];
00069         }
00070     }
00071     return NULL;
00072 }
00073
00075 score_t InputList::SpearmanRho(class InputList * in) {
00076     class InputItem * i1, * i2;
00077     int32_t diff = 0.0;
00078     rank_t sum = 0, i = 0, j = 0;
00079     score_t rho = 0.0;
00080
00081     double denom = pow(this->num_items, 3) - this->num_items;
00082
00083     for (i = 0; i < this->num_items; i++) {
00084         i1 = this->items[i];
00085
00086         printf("Searching for %s...  ", i1->get_code());
00087
00088         for (j = 0; j < in->get_num_items(); j++) {
00089             i2 = in->get_item(j);
00090
00091             if ( strcmp( i1->get_code(), i2->get_code() ) == 0) {
00092                 diff = i1->get_rank() - i2->get_rank();
00093                 sum += pow (diff, 2.0 );
00094 //              printf("Found!  (%d vs %d) ---> Sum = %d\n", i1->get_rank(), i2->get_rank(), sum);
00095                 break;
00096             }
00097         }
00098     }
00099
00100     rho = 1.0 - 6.0 * sum / denom;
00101     return rho;
00102 }
00103
00106 void InputList::sort_by_score() {
00107     rank_t i = 0;
00108     score_t sum = 0.0, mean = 0.0;
00109
00110     if (this->num_items > 0) {
00111
00112         qsort(this->items, this->num_items, sizeof(class InputItem *), &InputList::cmp_score);
00113
00115         this->stats->set_min_val( this->items[this->num_items - 1]->get_inscore() );
00116         this->stats->set_max_val( this->items[0]->get_inscore() );
00117
00119         for (i = 0; i < this->num_items; i++) {
00120             this->items[i]->set_rank(i + 1);
00121             sum += this->items[i]->get_inscore();
00122         }
00123         mean = sum / (score_t)this->num_items;
00124         this->stats->set_mean_val( mean );
00125
00127         sum = 0.0;
00128         for (i = 0; i < this->num_items; i++) {
00129             sum += (this->items[i]->get_inscore() - mean) * (this->items[i]->get_inscore() - mean);
00130         }
00131
00132         this->stats->set_std_val( sqrt(sum / this->num_items) );
00133     }
00134 }
00135
00137 void InputList::display() {
00138     this->voter->display();
00139
00140     printf("\tNum Items:  %d\n\tCutoff Point:  %d\n", this->num_items, this->cutoff);
00141     for (uint32_t i = 0; i < this->num_items; i++) {
00142         printf("\t%d:  ", i);
00143         this->items[i]->display();
00144     }
00145 }
00146
00148 void InputList::set_id(uint32_t v) { this->id = v; }
00149 void InputList::set_voter_weight(double v) { this->voter->set_weight(v); }
00150 void InputList::set_cutoff(rank_t v) { this->cutoff = v; }
00151
00153 uint32_t InputList::get_id() { return this->id; }
00154 class Voter * InputList::get_voter() { return this->voter; }
```

```
00155 rank_t InputList::get_num_items() { return this->num_items; }
00156 class InputItem * InputList::get_item(rank_t i) { return this->items[i]; }
00157 rank_t InputList::get_cutoff() { return this->cutoff; }
00158
00159 score_t InputList::get_min_score() { return this->stats->get_min_val(); }
00160 score_t InputList::get_max_score() { return this->stats->get_max_val(); }
00161 score_t InputList::get_mean_score() { return this->stats->get_mean_val(); }
00162 score_t InputList::get_std_score() { return this->stats->get_std_val(); }
```

6.32 FLAGR/src/InputList.h File Reference

Classes

- class InputList

6.33 InputList.h

Go to the documentation of this file.

```
00001 #ifndef INPUTLIST_H
00002 #define INPUTLIST_H
00003
00004
00005 class InputList {
00006     private:
00007         uint32_t id;
00008         class Voter * voter;
00009
00010         rank_t num_items;
00011         rank_t num_alloc_items;
00012         rank_t cutoff;
00013
00014         class SimpleScoreStats * stats;
00015
00016         class InputItem ** items;
00017
00018     private:
00019         static int cmp_score(const void *a, const void *b) {
00020             class InputItem * x = *(class InputItem **)a;
00021             class InputItem * y = *(class InputItem **)b;
00022
00023             return y->get_inscore() - x->get_inscore();
00024         }
00025
00026     public:
00027         InputList();
00028         InputList(uint32_t, char *, score_t);
00029         ~InputList();
00030
00031         void insert_item(char *, rank_t, score_t);
00032         void replace_item(char *, rank_t, score_t);
00033         class InputItem * search_item(char *);
00034         void display();
00035         void sort_by_score();
00036
00037         score_t SpearmanRho(class InputList *);
00038
00040         void set_id(uint32_t);
00041         void set_voter_weight(double);
00042         void set_cutoff(rank_t);
00043
00045         uint32_t get_id();
00046         class Voter * get_voter();
00047         rank_t get_num_items();
00048         rank_t get_cutoff();
00049         class InputItem * get_item(rank_t);
00050
00051         score_t get_min_score();
00052         score_t get_max_score();
00053         score_t get_mean_score();
00054         score_t get_std_score();
00055 };
00056
00057 #endif // INPUTLIST_H
```

6.34 FLAGR/src/InputParams.cpp File Reference

```
#include "InputParams.h"
```

6.35 InputParams.cpp

Go to the documentation of this file.

```
00001 #include "InputParams.h"
00002
00004 InputParams::InputParams() :
00005         input_file(NULL),
00006         rels_file(NULL),
00007         output_file(NULL),
00008         eval_file(NULL),
00009         random_string(NULL),
00010         aggregation_method(0),
00011         correlation_method(0),
00012         weights_normalization(0),
00013         max_iterations(0),
00014         max_list_items(0),
00015         eval_points(0),
00016         list_pruning(false),
00017         exact(false),
00018         convergence_precision(0.0),
00019         alpha(0.0),
00020         beta(0.0),
00021         gamma(0.0),
00022         delta1(0.0),
00023         delta2(0.0),
00024         c1(0.0),
00025         c2(0.0),
00026         pref_thr(0.0),
00027         veto_thr(0.0),
00028         conc_thr(0.0),
00029         disc_thr(0.0) { }
00030
00032 InputParams::InputParams(struct UserParams uParams) :
00033         input_file(NULL),
00034         rels_file(NULL),
00035         output_file(NULL),
00036         eval_file(NULL),
00037         random_string(NULL),
00038         aggregation_method(uParams.rank_aggregation_method),
00039         correlation_method(uParams.distance),
00040         weights_normalization(uParams.weight_normalization),
00041         max_iterations(uParams.max_iter),
00042         max_list_items(0),
00043         eval_points(uParams.eval_points),
00044         list_pruning(uParams.prune),
00045         exact(uParams.exact),
00046         convergence_precision(uParams.tol),
00047         alpha(uParams.alpha),
00048         beta(uParams.beta),
00049         gamma(uParams.gamma),
00050         delta1(uParams.delta1),
00051         delta2(uParams.delta2),
00052         c1(uParams.c1),
00053         c2(uParams.c2),
00054         pref_thr(uParams.pref_thr),
00055         veto_thr(uParams.veto_thr),
00056         conc_thr(uParams.conc_thr),
00057         disc_thr(uParams.disc_thr) {
00058
00059             this->set_input_file(uParams.input_file);
00060             if (uParams.rels_file) {
00061                 this->set_rels_file(uParams.rels_file);
00062             }
00063
00064             this->set_random_string(uParams.random_string);
00065             this->set_output_files(uParams.output_dir);
00066         }
00067
00069 InputParams::~InputParams() {
00070     if (this->input_file) { delete [] this->input_file; }
00071     if (this->rels_file) { delete [] this->rels_file; }
00072     if (this->output_file) { delete [] this->output_file; }
00073     if (this->eval_file) { delete [] this->eval_file; }
```

```
00074     if (this->random_string) { delete [] this->random_string; }
00075 }
00076
00078 void InputParams::display() {
00079     printf("FLAGR execution parameters:\n");
00080     if (input_file)     { printf("\tInput file:                 %s\n", this->input_file); }
00081                   else  { printf("\tInput file:                 [not set]\n"); }
00082
00083     if (rels_file)      { printf("\tQ-Rels file:                %s\n", this->rels_file); }
00084                   else  { printf("\tQ-Rels file:                [not set]\n"); }
00085
00086     if (output_file)    { printf("\tOutput file:                %s\n", this->output_file); }
00087                   else  { printf("\tOutput file:                [not set]\n"); }
00088
00089     if (eval_file)      { printf("\tEvaluation file:          %s\n", this->eval_file); }
00090                   else  { printf("\tEvaluation file:          [not set]\n"); }
00091
00092     if (random_string)  { printf("\tRandom string:              %s\n", this->random_string); }
00093                   else  { printf("\tRandom string:              [not set]\n"); }
00094
00095     printf("\tAggregation method:  %d\n",    this->aggregation_method);
00096     printf("\tDistance measure:      %d\n",    this->correlation_method);
00097     printf("\tVoter weights norm:  %d\n",    this->weights_normalization);
00098     printf("\tMax iterations:          %d\n",    this->max_iterations);
00099     printf("\tMax list items:          %d\n",    this->max_list_items);
00100     printf("\tEvaluation points:    %d\n",    this->eval_points);
00101
00102     printf("\nAlgorithm hyper-parameters:\n");
00103     printf("\tList pruning:              %d\n",    this->list_pruning);
00104     printf("\tExact computation:    %d\n",    this->exact);
00105     printf("\tConvergence precis:  %7.6f\n", this->convergence_precision);
00106     printf("\talpha:                              %4.3f\n", this->alpha);
00107     printf("\tbeta:                               %4.3f\n", this->beta);
00108     printf("\tc_1:                                %4.3f\n", this->c1);
00109     printf("\tc_2:                                %4.3f\n", this->c2);
00110     printf("\tgamma:                            %4.3f\n", this->gamma);
00111     printf("\tdelta_1:                         %4.3f\n", this->delta1);
00112     printf("\tdelta_2:                         %4.3f\n", this->delta2);
00113     printf("\tPreference thresh:    %4.3f\n", this->pref_thr);
00114     printf("\tVeto threshold:          %4.3f\n", this->veto_thr);
00115     printf("\tConcordance thresh:  %4.3f\n", this->conc_thr);
00116     printf("\tDiscordance thresh:  %4.3f\n", this->disc_thr);
00117     printf("\n"); fflush(NULL);
00118 }
00119
00121 void InputParams::generate_random_string(size_t size) {
00122
00123     const char charset[] = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJK0123456789";
00124     this->random_string = new char[size + 1];
00125
00126     if (size) {
00127         --size;
00128         for (size_t n = 0; n < size; n++) {
00129             int key = rand() % (int) (sizeof charset - 1);
00130             this->random_string[n] = charset[key];
00131         }
00132         this->random_string[size] = '\0';
00133     }
00134 }
00135
00137 void InputParams::set_output_files(char * out_dir) {
00138     if (!this->random_string) {
00139         this->generate_random_string(16);
00140     }
00141
00142     this->output_file = new char[1024];
00143     sprintf(this->output_file, "%s/out_%s.csv", out_dir, this->random_string);
00144
00145     if (this->rels_file) {
00146         this->eval_file = new char[1024];
00147         sprintf(this->eval_file, "%s/eval_%s.csv", out_dir, this->random_string);
00148     }
00149
00150     FILE * fp = NULL;
00151     fp = fopen(this->output_file, "w+");
00152     if (fp) {
00153         fprintf(fp, "Query,Voter,ItemID,Score\n");
00154         fclose(fp);
00155     }
00156
00157     fp = fopen(this->eval_file, "w+");
00158     if (fp) { fclose(fp); }
00159 }
00160
00161
00163 char * InputParams::get_input_file() { return this->input_file; }
00164 char * InputParams::get_rels_file() { return this->rels_file; }
```

```
00165 char * InputParams::get_output_file() { return this->output_file; }
00166 char * InputParams::get_eval_file() { return this->eval_file; }
00167 char * InputParams::get_random_string() { return this->random_string; }
00168
00169 uint32_t InputParams::get_aggregation_method() { return this->aggregation_method; }
00170 uint32_t InputParams::get_correlation_method() { return this->correlation_method; }
00171 uint32_t InputParams::get_weights_normalization() { return this->weights_normalization; }
00172 int32_t InputParams::get_max_iterations() { return this->max_iterations; }
00173 uint32_t InputParams::get_max_list_items() { return this->max_list_items; }
00174 rank_t InputParams::get_eval_points() { return this->eval_points; }
00175 bool InputParams::get_list_pruning() { return this->list_pruning; }
00176 bool InputParams::get_exact() { return this->exact; }
00177
00178 score_t InputParams::get_convergence_precision() { return this->convergence_precision; }
00179 score_t InputParams::get_alpha() { return this->alpha; }
00180 score_t InputParams::get_beta() { return this->beta; }
00181 score_t InputParams::get_gamma() { return this->gamma; }
00182 score_t InputParams::get_delta1() { return this->delta1; }
00183 score_t InputParams::get_delta2() { return this->delta2; }
00184 score_t InputParams::get_c1() { return this->c1; }
00185 score_t InputParams::get_c2() { return this->c2; }
00186 score_t InputParams::get_pref_thr() { return this->pref_thr; }
00187 score_t InputParams::get_veto_thr() { return this->veto_thr; }
00188 score_t InputParams::get_conc_thr() { return this->conc_thr; }
00189 score_t InputParams::get_disc_thr() { return this->disc_thr; }
00190
00191
00194 void InputParams::set_input_file(char * v) {
00195     this->input_file = new char[strlen(v) + 1];
00196     strcpy(this->input_file, v);
00197 }
00198
00199 void InputParams::set_rels_file(char * v) {
00200     this->rels_file = new char[strlen(v) + 1];
00201     strcpy(this->rels_file, v);
00202 }
00203
00204 void InputParams::set_output_file(const char * v) {
00205     this->output_file = new char[strlen(v) + 1];
00206     strcpy(this->output_file, v);
00207 }
00208
00209 void InputParams::set_eval_file(const char * v) {
00210     this->eval_file = new char[strlen(v) + 1];
00211     strcpy(this->eval_file, v);
00212 }
00213
00214 void InputParams::set_random_string(const char * v) {
00215     this->random_string = new char[strlen(v) + 1];
00216     strcpy(this->random_string, v);
00217 }
00218
00219 void InputParams::set_aggregation_method(uint32_t v) { this->aggregation_method = v; }
00220 void InputParams::set_correlation_method(uint32_t v) { this->correlation_method = v; }
00221 void InputParams::set_weights_normalization(uint32_t v) { this->weights_normalization = v; }
00222 void InputParams::set_max_iterations(int32_t v) { this->max_iterations = v; }
00223 void InputParams::set_max_list_items(uint32_t v) { this->max_list_items = v; }
00224 void InputParams::set_eval_points(rank_t v) { this->eval_points = v; }
00225 void InputParams::set_list_pruning(bool v) { this->list_pruning = v; }
00226
00227 void InputParams::set_convergence_precision(score_t v) { this->convergence_precision = v; }
00228 void InputParams::set_alpha(score_t v) { this->alpha = v; }
00229 void InputParams::set_beta(score_t v) { this->beta = v; }
00230 void InputParams::set_gamma(score_t v) { this->gamma = v; }
00231 void InputParams::set_delta1(score_t v) { this->delta1 = v; }
00232 void InputParams::set_delta2(score_t v) { this->delta2 = v; }
00233 void InputParams::set_c1(score_t v) { this->c1 = v; }
00234 void InputParams::set_c2(score_t v) { this->c2 = v; }
00235 void InputParams::set_pref_thr(score_t v) { this->pref_thr = v; }
00236 void InputParams::set_veto_thr(score_t v) { this->veto_thr = v; }
00237 void InputParams::set_conc_thr(score_t v) { this->conc_thr = v; }
00238 void InputParams::set_disc_thr(score_t v) { this->disc_thr = v; }
```

6.36 FLAGR/src/InputParams.h File Reference

Classes

- struct UserParams

  External user parameters are passed to FLAGR via this UserParams structure.
- class InputParams

6.37 InputParams.h

Go to the documentation of this file.

```
00001 #ifndef INPUTPARAMS_H
00002 #define INPUTPARAMS_H
00003
00005 struct UserParams {
00006     char * input_file = NULL;
00007     char * rels_file = NULL;
00008     char * random_string = NULL;
00009     char * output_dir = NULL;
00010
00011     int eval_points = 0;
00012
00013     int rank_aggregation_method = 0;
00014     int weight_normalization = 0;
00015     int distance = 0;
00016
00017     float tol = 0.0;
00018     int max_iter = 0;
00019     bool prune = false;
00020     bool exact = false;
00021
00022     score_t pref_thr = 0.0;
00023     score_t veto_thr = 0.0;
00024     score_t conc_thr = 0.0;
00025     score_t disc_thr = 0.0;
00026
00027     score_t alpha = 0.0;
00028     score_t beta = 0.0;
00029     score_t gamma = 0.0;
00030     score_t delta1 = 0.0;
00031     score_t delta2 = 0.0;
00032     score_t c1 = 0.0;
00033     score_t c2 = 0.0;
00034 };
00035
00036
00073
00074
00081
00087
00088 class InputParams {
00089     private:
00090         char * input_file;
00091         char * rels_file;
00092         char * output_file;
00093         char * eval_file;
00094         char * random_string;
00095
00096         uint32_t aggregation_method;
00097         uint32_t correlation_method;
00098         uint32_t weights_normalization;
00099         int32_t max_iterations;
00100         uint32_t max_list_items;
00101         rank_t eval_points;
00102         bool list_pruning;
00103         bool exact;
00104
00105         score_t convergence_precision;
00106         score_t alpha;
00107         score_t beta;
00108         score_t gamma;
00109         score_t delta1;
00110         score_t delta2;
00111         score_t c1;
00112         score_t c2;
00113         score_t pref_thr;
00114         score_t veto_thr;
00115         score_t conc_thr;
00116         score_t disc_thr;
00117
00118     private:
00119         void generate_random_string(size_t);
00120
00121     public:
00122         InputParams();
00123         InputParams(struct UserParams);
00124         ~InputParams();
00125
00126         void set_output_files(char *);
00127         void display();
00128
00130         char * get_input_file();
00131         char * get_rels_file();
```

```
00132         char * get_output_file();
00133         char * get_eval_file();
00134         char * get_random_string();
00135 
00136         uint32_t get_aggregation_method();
00137         uint32_t get_correlation_method();
00138         uint32_t get_weights_normalization();
00139         int32_t get_max_iterations();
00140         int32_t get_iterations();
00141         uint32_t get_max_list_items();
00142         rank_t get_eval_points();
00143         bool get_list_pruning();
00144         bool get_exact();
00145 
00146         score_t get_convergence_precision();
00147         score_t get_alpha();
00148         score_t get_beta();
00149         score_t get_gamma();
00150         score_t get_delta1();
00151         score_t get_delta2();
00152         score_t get_c1();
00153         score_t get_c2();
00154         score_t get_pref_thr();
00155         score_t get_veto_thr();
00156         score_t get_conc_thr();
00157         score_t get_disc_thr();
00158 
00160         void set_input_file(char *);
00161         void set_rels_file(char *);
00162         void set_output_file(const char *);
00163         void set_eval_file(const char *);
00164         void set_random_string(const char *);
00165 
00166         void set_aggregation_method(uint32_t);
00167         void set_correlation_method(uint32_t);
00168         void set_weights_normalization(uint32_t);
00169         void set_max_iterations(int32_t);
00170         void set_iterations(int32_t);
00171         void set_max_list_items(uint32_t);
00172         void set_eval_points(rank_t);
00173         void set_list_pruning(bool);
00174 
00175         void set_convergence_precision(score_t);
00176         void set_alpha(score_t);
00177         void set_beta(score_t);
00178         void set_gamma(score_t);
00179         void set_delta1(score_t);
00180         void set_delta2(score_t);
00181         void set_c1(score_t);
00182         void set_c2(score_t);
00183         void set_pref_thr(score_t);
00184         void set_veto_thr(score_t);
00185         void set_conc_thr(score_t);
00186         void set_disc_thr(score_t);
00187 };
00188 
00189 #endif // INPUTPARAMS_H
```

6.38 FLAGR/src/MergedItem.cpp File Reference

```
#include "MergedItem.h"
#include "ram/tools/BetaDistribution.cpp"
```

6.39 MergedItem.cpp

Go to the documentation of this file.

```
00001 #include "MergedItem.h"
00002 #include "ram/tools/BetaDistribution.cpp"
00003 
00005 MergedItem::MergedItem() :
00006     final_score (0.0),
00007     final_ranking(0),
00008     num_rankings(0),
```

```
00009     num_alloc_rankings(0),
00010     rankings(nullptr),
00011     next(nullptr) {
00012 }
00013
00015 MergedItem::MergedItem(class MergedItem * in) {
00016     class Ranking * r = NULL;
00017
00018     this->code = new char[strlen(in->get_code()) + 1];
00019     strcpy(this->code, in->get_code());
00020
00021     this->final_score = in->get_final_score();
00022     this->final_ranking = in->get_final_ranking();
00023     this->num_rankings = in->get_num_rankings();
00024     this->num_alloc_rankings = in->get_num_alloc_rankings();
00025
00026     this->rankings = new Ranking * [this->num_alloc_rankings];
00027     for (uint32_t i = 0; i < this->num_alloc_rankings; i++) {
00028         r = in->get_ranking(i);
00029         this->rankings[i] = new Ranking( r->get_input_list(), r->get_rank(), r->get_score() );
00030     }
00031     this->next = NULL;
00032 }
00033
00035 MergedItem::MergedItem(char * c, rank_t r, uint32_t nal, class InputList ** l) :  InputItem(c, r, 0.0)
    {
00036     this->final_score = 0.0;
00037     this->final_ranking = 0;
00038     this->num_rankings = 0;
00039     this->num_alloc_rankings = nal;
00040
00041     this->rankings = new Ranking * [this->num_alloc_rankings];
00042     for (uint32_t i = 0; i < this->num_alloc_rankings; i++) {
00043         this->rankings[i] = new Ranking(l[i], NOT_RANKED_ITEM_RANK, 0.0);
00044     }
00045     this->next = NULL;
00046 }
00047
00049 MergedItem::~MergedItem() {
00050     if (this->rankings) {
00051         for (rank_t i = 0; i < this->num_alloc_rankings; i++) {
00052             if (this->rankings[i]) {
00053                 delete this->rankings[i];
00054             }
00055         }
00056         delete [] this->rankings;
00057     }
00058 }
00059
00060
00062 void MergedItem::insert_ranking(class InputList * l, rank_t r, score_t s) {
00063     this->rankings[ l->get_id() ]->set_input_list (l);
00064     this->rankings[ l->get_id() ]->set_rank(r);
00065     this->rankings[ l->get_id() ]->set_score(s);
00066     this->num_rankings++;
00067 }
00068
00069
00071 void MergedItem::display() {
00072     class Ranking * r;
00073     printf("Item:  %s was found in %d input lists, Score:  %E:\n",
00074         this->code, this->num_rankings, this->final_score);
00075
00076     for (uint32_t i = 0; i < this->num_alloc_rankings; i++) {
00077         r = this->rankings[i];
00078         if (r->get_input_list()) {
00079             r->display();
00080         }
00081     }
00082     printf("\n");
00083 }
00084
00086 void MergedItem::sort_rankings_by_score() {
00087     qsort(this->rankings, this->num_alloc_rankings, sizeof(class Ranking *), &MergedItem::cmp_score);
00088 }
00089
00091 int MergedItem::cmp_score(const void *a, const void *b) {
00092     class Ranking * x = * (class Ranking **)a;
00093     class Ranking * y = * (class Ranking **)b;
00094
00095     if (x->get_score() > y->get_score()) {
00096         return 1;
00097     } else {
00098         return -1;
00099     }
00100 }
00101
```

```
00103 void MergedItem::compute_beta_values() {
00104     class Ranking * r;
00105     double p = 0.0;
00106 
00107     for (uint32_t i = 0; i < this->num_alloc_rankings; i++) {
00108         r = this->rankings[i];
00109 
00110         p = pbeta(r->get_score(), i + 1, this->num_alloc_rankings - i);
00111 
00112         r->set_score( p );
00113     }
00114 }
00115 
00117 void MergedItem::set_final_score(score_t v) { this->final_score = v; }
00118 void MergedItem::set_final_ranking(rank_t v) { this->final_ranking = v; }
00119 void MergedItem::set_next(class MergedItem * v) { this->next = v; }
00120 
00122 score_t MergedItem::get_final_score() { return this->final_score; }
00123 rank_t MergedItem::get_final_ranking() { return this->final_ranking; }
00124 uint32_t MergedItem::get_num_rankings() { return this->num_rankings; }
00125 uint32_t MergedItem::get_num_alloc_rankings() { return this->num_alloc_rankings; }
00126 class MergedItem * MergedItem::get_next() { return this->next; }
00127 class Ranking * MergedItem::get_ranking(uint32_t i) { return this->rankings[i]; }
```

6.40 FLAGR/src/MergedItem.h File Reference

Classes

- class MergedItem

6.41 MergedItem.h

Go to the documentation of this file.

```
00001 #ifndef MERGEDITEM_H
00002 #define MERGEDITEM_H
00003 
00004 class MergedItem :  public InputItem {
00005     private:
00006         score_t final_score;
00007         rank_t final_ranking;
00008         uint32_t num_rankings;
00009         uint32_t num_alloc_rankings;
00010 
00011         class Ranking ** rankings;
00012 
00013         class MergedItem * next;
00014 
00015     private:
00017         static int cmp_score(const void *a, const void *b);
00018 
00019     public:
00020         MergedItem();
00021         MergedItem(class MergedItem *);
00022         MergedItem(char *, rank_t, uint32_t, class InputList **);
00023         ~MergedItem();
00024 
00025         void insert_ranking(class InputList *, rank_t, score_t);
00026         void sort_rankings_by_score();
00027         void compute_beta_values();
00028         void display();
00029 
00031         void set_final_score(score_t);
00032         void set_final_ranking(rank_t);
00033         void set_next(class MergedItem *);
00034 
00036         score_t get_final_score();
00037         rank_t get_final_ranking();
00038         uint32_t get_num_rankings();
00039         uint32_t get_num_alloc_rankings();
00040         class MergedItem * get_next();
00041         class Ranking * get_ranking(uint32_t);
00042 };
00043 
00044 #endif // MERGEDITEM_H
```

6.42 FLAGR/src/MergedList.cpp File Reference

```
#include "MergedList.h"
#include "ram/CombSUM.cpp"
#include "ram/CombMNZ.cpp"
#include "ram/CondorcetWinners.cpp"
#include "ram/CopelandWinners.cpp"
#include "ram/OutrankingApproach.cpp"
#include "ram/KemenyOptimal.cpp"
#include "ram/DIBRA.cpp"
#include "ram/PrefRel.cpp"
#include "ram/Agglomerative.cpp"
#include "ram/MC.cpp"
#include "ram/RobustRA.cpp"
#include "ram/CustomMethods.cpp"
```

6.43 MergedList.cpp

Go to the documentation of this file.

```
00001 #include "MergedList.h"
00002
00003 #include "ram/CombSUM.cpp"
00004 #include "ram/CombMNZ.cpp"
00005 #include "ram/CondorcetWinners.cpp"
00006 #include "ram/CopelandWinners.cpp"
00007 #include "ram/OutrankingApproach.cpp"
00008 #include "ram/KemenyOptimal.cpp"
00009 #include "ram/DIBRA.cpp"
00010 #include "ram/PrefRel.cpp"
00011 #include "ram/Agglomerative.cpp"
00012 #include "ram/MC.cpp"
00013 #include "ram/RobustRA.cpp"
00014 #include "ram/CustomMethods.cpp"
00015
00016
00018 MergedList::MergedList() :
00019         num_input_lists(0),
00020         hash_table(NULL),
00021         item_list(NULL),
00022         mask(0),
00023         num_slots(0),
00024         num_nodes(0),
00025         num_chains(0),
00026         weight(0.0),
00027         log10s{0.0} { }
00028
00030 MergedList::MergedList(uint32_t size, uint32_t n) :
00031         num_input_lists(n),
00032         hash_table(new MergedItem * [size]),
00033         item_list(NULL),
00034         mask(size - 1),
00035         num_slots(size),
00036         num_nodes(0),
00037         num_chains(0),
00038         weight(0.0),
00039         log10s{0.0} {
00040
00041             for (uint32_t i = 0; i < size; i++) {
00042                 this->hash_table[i] = NULL;
00043             }
00044
00045             this->log10s[0] = 0;
00046             for (uint32_t i = 1; i < 100000; i++) {
00047                 this->log10s[i] = log10( (double)i );
00048             }
00049 }
00050
00053 MergedList::MergedList(class InputList ** inlists, uint32_t nlists, uint32_t m) {
00054     uint32_t size = 1024;
00055     class InputItem * elem = NULL;
00056     rank_t nitems = inlists[m]->get_num_items();
```

```
00057
00058     score_t score = 0;
00059
00060     this->weight = inlists[m]->get_voter()->get_weight();
00061     this->num_input_lists = nlists;
00062     this->num_nodes = 0;
00063     this->num_chains = 0;
00064     this->num_slots = size;
00065     this->mask = size - 1;
00066     this->item_list = NULL;
00067
00068     this->hash_table = new MergedItem * [size];
00069
00070     for (uint32_t i = 0; i < size; i++) {
00071         this->hash_table[i] = NULL;
00072     }
00073
00074 //  printf("\ncreating list %d...\n", m);fflush((NULL));
00078     for (rank_t i = 0; i < inlists[m]->get_num_items(); i++) {
00079         elem = inlists[m]->get_item(i);
00080
00082         score = 0.5 * (int32_t)(nitems * nitems - 2 * nitems * (elem->get_rank() - 1) - nitems);
00083
00084 //      printf("\tinserting item %d :  %s (rank %d, score %+5.2f)...\n", i, elem->get_code(),
00085 //              elem->get_rank(), score); fflush((NULL));
00086
00087         this->insert(elem, m, inlists);
00088         this->update_weight(elem->get_code(), score);
00089     }
00090
00091     this->log10s[0] = 0;
00092     for (uint32_t i = 1; i < 100000; i++) {
00093         this->log10s[i] = log10( (double)i );
00094     }
00095
00096     this->convert_to_array();
00097     qsort(this->item_list, this->num_nodes, sizeof(class MergedItem *), &MergedList::cmp_score_desc);
00098 }
00099
00101 MergedList::~MergedList() {
00102     this->clear_contents();
00103 }
00104
00106 void MergedList::insert(class InputItem * n, uint32_t x, class InputList ** l) {
00108     uint32_t HashValue = this->djb2(n->get_code()) & this->mask;
00109
00111     if (this->hash_table[HashValue] != NULL) {
00112         class MergedItem * q;
00113
00115         for (q = this->hash_table[HashValue]; q != NULL; q = q->get_next()) {
00116             if (strcmp(q->get_code(), n->get_code()) == 0) {
00117
00118                 q->insert_ranking( l[x], n->get_rank(), n->get_inscore() );
00119
00120                 return;
00121             }
00122         }
00123     } else {
00124         this->num_chains++;
00125     }
00126
00127     this->num_nodes++;
00128
00130     class MergedItem * record = new MergedItem(n->get_code(), n->get_rank(), this->num_input_lists,
      l);
00131     record->insert_ranking( l[x], n->get_rank(), n->get_inscore() );
00132
00134     record->set_next(this->hash_table[HashValue]);
00135     this->hash_table[HashValue] = record;
00136 }
00137
00139 void MergedList::update_weight(char * code, score_t w) {
00141     uint32_t HashValue = this->djb2(code) & this->mask;
00142
00144     if (this->hash_table[HashValue] != NULL) {
00145         class MergedItem * q;
00146
00148         for (q = this->hash_table[HashValue]; q != NULL; q = q->get_next()) {
00149             if (strcmp(q->get_code(), code) == 0) {
00150
00151                 q->set_final_score( q->get_final_score() + w );
00152
00153                 return;
00154             }
00155         }
00156     }
00157 }
```

```
00158
00159
00161 void MergedList::display() {
00162     class MergedItem * q;
00163     for (uint32_t i = 0; i < this->num_slots; i++) {
00164         if (this->hash_table[i] != NULL) {
00165             for (q = this->hash_table[i]; q != NULL; q = q->get_next()) {
00166                 q->display();
00167 //                getchar();
00168             }
00169         }
00170     }
00171 }
00172
00173
00175 void MergedList::display_list() {
00176     for (rank_t i = 0; i < this->num_nodes; i++) {
00177         this->item_list[i]->display();
00178 //      getchar();
00179     }
00180 }
00181
00182
00184 void MergedList::write_to_CSV(char * topic, class InputParams * params) {
00185     FILE * fp = fopen(params->get_output_file(), "a+");
00186     if (fp) {
00187         for (rank_t i = 0; i < this->num_nodes; i++) {
00188             fprintf(fp, "%s,PyFLAGR,%s,%d,%10.6f\n", topic, this->item_list[i]->get_code(), i+1,
00189                     this->item_list[i]->get_final_score());
00190         }
00191         fclose(fp);
00192     }
00193 }
00194
00195
00197 void MergedList::convert_to_array() {
00198     rank_t x = 0;
00199
00200     this->item_list = new MergedItem * [this->num_nodes];
00201
00202     class MergedItem * q;
00203     for (uint32_t i = 0; i < this->num_slots; i++) {
00204         if (this->hash_table[i] != NULL) {
00205             for (q = this->hash_table[i]; q != NULL; q = q->get_next()) {
00206                 this->item_list[x++] = q;
00207             }
00208         }
00209     }
00210 }
00211
00213 void MergedList::reset_scores() {
00214     for (rank_t i = 0; i < this->num_nodes; i++) {
00215         this->item_list[i]->set_final_score(0.0);
00216     }
00217 }
00218
00220 rank_t MergedList::get_item_rank(char *c) {
00221     uint32_t HashValue = this->djb2(c) & this->mask;
00222
00223     if (this->hash_table[HashValue] != NULL) {
00224         class MergedItem * q;
00225
00226         for (q = this->hash_table[HashValue]; q != NULL; q = q->get_next()) {
00227             if (strcmp(q->get_code(), c) == 0) {
00228                 return q->get_final_ranking();
00229             }
00230         }
00231     }
00232     return NOT_RANKED_ITEM_RANK;
00233 }
00234
00236 void MergedList::clear_contents() {
00237     class MergedItem * q;
00238
00239     if (this->hash_table) {
00240         for (uint32_t i = 0; i < this->num_slots; i++) {
00241             while (this->hash_table[i] != NULL) {
00242                 q = this->hash_table[i]->get_next();
00243                 delete this->hash_table[i];
00244                 this->hash_table[i] = q;
00245             }
00246         }
00247
00248         delete [] this->hash_table;
00249         this->hash_table = NULL;
00250     }
00251
```

```
00252     if (this->item_list) {
00253         delete [] this->item_list;
00254         this->item_list = NULL;
00255     }
00256 
00257     this->num_nodes = 0;
00258     this->num_chains = 0;
00259 }
00260 
00262 void MergedList::rebuild(class InputList ** inlists) {
00263     uint32_t k = 0, l = 0;
00264 
00265     this->clear_contents();
00266     this->hash_table = new MergedItem * [this->num_slots];
00267 
00268     for (uint32_t i = 0; i < this->num_slots; i++) {
00269         this->hash_table[i] = NULL;
00270     }
00271 
00272     this->num_nodes = 0;
00273     this->num_chains = 0;
00274     this->item_list = NULL;
00275 
00276     for (l = 0; l < this->num_input_lists; l++) {
00277         for (k = 0; k < inlists[l]->get_cutoff(); k++) {
00278             this->insert(inlists[l]->get_item(k), l, inlists);
00279         }
00280     }
00281 
00282     this->convert_to_array();
00283     this->reset_scores();
00284 }
00285 
00286 
00291 double MergedList::SpearmanRho(class InputList * in) {
00292     class MergedItem * q;
00293     double rho = 0.0, sum = 0.0;
00294     rank_t n = this->num_nodes;
00295 
00296     double denom = pow(n, 3) - n;
00297 
00298     for (rank_t i = 0; i < n; i++) {
00299         q = this->item_list[i];
00300 
00301         for (uint32_t j = 0; j < this->num_input_lists; j++) {
00302             if(q->get_ranking(j)->get_input_list() == in && q->get_ranking(j)->get_rank() !=
    NOT_RANKED_ITEM_RANK) {
00303                 sum += pow(q->get_ranking(j)->get_rank() - (i + 1.0), 2.0);
00304 //              printf("Item %d of Merged List, Score:  %5.3f was ranked in place %d in list %d -
    Sum=%5.3f\n",
00305 //                   i, q->get_score(), q->get_ranking(j)->Rank, j, sum);
00306             }
00307         }
00308     }
00309 
00310     rho = 1.0 - 6.0 * sum / denom;
00311 //  printf("(%d-%d) - rho=%5.3f - sum:%5.3f\n", this->num_nodes, in->get_num_items(), rho, sum);
00312     return rho;
00313 }
00314 
00315 double MergedList::KendallsTau(uint32_t z, class InputList * in) {
00316     rank_t a = 0, b = 0, n = in->get_cutoff(), concordant = 0, discordant = 0;
00317     double tau = 0.0, denom = n * (n - 1.0) / 2.0;
00318 
00319     for (rank_t i = 0; i < n; i++) {
00320         a = this->get_item_rank( in->get_item(i)->get_code() );
00321 
00322         for (rank_t j = i + 1; j < n; j++) {
00323             b = this->get_item_rank( in->get_item(j)->get_code() );
00324 //          printf("Comparing pair (%d:%s, %d:%s) -> (%d, %d) - ", i, in->get_item(i)->get_code(), j,
    in->get_item(j)->get_code(), a, b);
00325             if (a < b) {
00326 //              printf("concordant\n");
00327                 concordant++;
00328             } else {
00329 //              printf("discordant\n");
00330                 discordant++;
00331             }
00332         }
00333     }
00334 
00335     tau = (double)(concordant) / denom;
00336 //  printf("Concordant :  %d - Discordant:  %d - Tau = %5.3f\n", concordant, discordant, tau);
00337     return tau;
00338 }
00339 
00340 double MergedList::CosineSimilarity(uint32_t z, class InputList * in) {
```

```
00341     double l_score = 0.0, r_score = 0.0, c_score = 0.0, csim = 0.0;
00342     rank_t R = in->get_cutoff(), L = this->num_nodes, l = 0, r = 0;
00343
00344     uint32_t scenario = 2;
00345
00346     for (r = 0; r < R; r++) {
00347         if (scenario == 1) { r_score += 1.0 / ( (r + 1.0) * (r + 1.0) ); } else
00348         if (scenario == 2) { r_score += (R - r + 1.0) * (R - r + 1.0); } else
00349         if (scenario == 3) { r_score += (double)(R * R) / (double)( (r + R) * (r + R) ); } else
00350         if (scenario == 4) { r_score += (r + 1.0) * (r + 1.0); } else
00351         if (scenario == 5) { r_score += 1.0 / ( (r + 1.0) * (r + 1.0) ); }
00352         if (scenario == 6) { r_score += 1.0 / ( (r + 1.0) * (r + 1.0) ); }
00353     }
00354
00355     for (l = 0; l < L; l++) {
00357         if (scenario == 1 || scenario == 2 || scenario == 3 || scenario == 4) {
00358             l_score += (l + 1.0) * (l + 1.0);
00359         } else if (scenario == 5) {
00360             l_score += this->log10s[10 + l] * this->log10s[10 + l];
00361         } else if (scenario == 6) {
00362             r = this->item_list[l]->get_ranking(z)->get_rank();
00363             if(r < in->get_cutoff()) {
00364                 l_score += this->log10s[10 + l] * this->log10s[10 + l];
00365             }
00366         }
00367     }
00368
00369     for (l = 0; l < L; l++) {
00370         r = this->item_list[l]->get_ranking(z)->get_rank();
00371
00372         if(r < in->get_cutoff()) {
00373             if (scenario == 1) { c_score += (l + 1.0) / (r + 1.0); } else
00374             if (scenario == 2) { c_score += (R - r + 1.0) * (l + 1.0); } else
00375             if (scenario == 3) { c_score += (double)(R) / ( (l + 1.0) * (r + R) ); } else
00376             if (scenario == 4) { c_score += (r + 1.0) * (l + 1.0); } else
00377             if (scenario == 5) { c_score += this->log10s[10 + l] / (r + 1.0); } else
00378             if (scenario == 6) { c_score += this->log10s[10 + l] / (r + 1.0); }
00379         }
00380     }
00381
00383     csim = c_score / (sqrt(r_score) * sqrt(l_score));
00384
00386 //  csim = c_score / (r_score + l_score - c_score);
00387
00389 //  csim = 2.0 * c_score / (r_score + l_score);
00390
00391 //  csim = c_score / (r_score * l_score);
00392
00393 //  printf("Csim:  %5.3f\n", csim);
00394     return 1.0 - csim;
00395 }
00396
00397
00399 double MergedList::ScaledFootruleDistance(uint32_t z, class InputList * in) {
00400     double d = 0.0, nd = 0.0;
00401     rank_t i = 0, r = 0, R = in->get_num_items(), L = this->num_nodes;
00402
00403     for (i = 0; i < L; i++) {
00404         r = this->item_list[i]->get_ranking(z)->get_rank();
00405
00406         if(r < in->get_cutoff()) {
00407             d += fabs( (double) i / L - (double) r / R );
00408
00409 //          printf("Item %d/%d of Merged List (Score:  %5.3f) ranked in place %d/%d in list %d (%s) -
      Sum=%5.3f\n",
00410 //              i, this->num_nodes, this->item_list[i]->get_score(), r, in->get_num_items(),
00411 //              z, in->get_voter()->get_name(), d); getchar();
00412         }
00413     }
00414
00415     nd = 2.00 * d / R;
00416
00417 //  if (nd > 1) { printf("gt > 1:  ==%d== d=%5.3f NormD=%5.3f", R, d, nd); getchar(); }
00418 //  printf("SFD: %5.3f (Norm SFD: %5.3f) - Items:  %d\n", d, nd, R); getchar();
00419
00420     return nd;
00421 }
00422
00424 double MergedList::LocalScaledFootruleDistance(uint32_t z, class InputList * in) {
00425     double d = 0.0, nd = 0.0, factor = 0.0;
00426 //  double log11 = log10(11.0), log22 = log2(2.2);
00427     rank_t i = 0, r = 0, R = in->get_num_items(), L = this->num_nodes;
00428
00429     for (i = 0; i < L; i++) {
00430         r = this->item_list[i]->get_ranking(z)->get_rank();
00431
00432         if(r < in->get_cutoff()) {
```

```
00433 //          factor = log10( 10.0 * (1.1 - (double) r / R ) ) / log11;  // good with GAMMA=4.5 (0.243 /
      0.230)
00434 //          factor = log10( 10.0 * (1.1 - (double) i / L ) ) / log11;
00435 //          factor = 1.0;
00436             factor = (double)R / (r + 1.0); // BEST with 1.0 <= GAMMA <= 1.5 ( 0.247-0.160 / 0.232)
00437 //          factor = (double)(R - r + 1.0) / (R + r + 1.0); // GOOD with 6.0 <= GAMMA <= 7.0
00438
00439             d += factor * ( (double) (r) / (R) - (double) (i) / (L));
00440
00441 //          printf("Item %d/%d of Merged List (Score:  %5.3f) was ranked in place %d/%d in list %d
      (%s) - Sum=%5.3f\n",
00442 //              i, this->num_nodes, this->item_list[i]->get_score(), r, in->get_num_items(),
00443 //              z, in->get_voter()->get_name(), d); getchar();
00444         }
00445     }
00446
00447     nd = 2.0 * fabs(d) / R;
00448 //  nd = fabs(d);
00449
00450 //  printf("gt > 1:  ==%d== d=%5.3f NormD=%5.3f", R, d, nd); getchar();
00451 //  printf("SFD: %5.3f (Norm SFD: %5.3f) - Items:  %d\n", d, nd, R); getchar();
00452
00453     return nd;
00454 }
00455
00457 uint64_t MergedList::factorial(uint32_t n) {
00458     if (n == 0) { return 1; }
00459     if (n > 20) {
00460         fprintf(stderr, "Cannot compute factorials of numbers greater than 20\n");
00461         exit(1);
00462     }
00463
00464     uint64_t f = 1.0;
00465     for (uint32_t i = 2; i <= n; i++) {
00466         f *= i;
00467     }
00468     return f;
00469 }
00470
00473 double MergedList::factorial(double n) {
00474     if (n < 0) {
00475         fprintf(stderr, "Undefined");
00476     }
00477
00478     if (n > 170) {
00479         fprintf(stderr, "Infinity");
00480     }
00481     return tgamma(n + 1);
00482 }
00483
00484 double * MergedList::precompute_170_factorials() {
00485     double * factorials = new double[171];
00486     factorials[0] = 0;
00487     for (uint16_t i = 1; i < 171; i++) {
00488         factorials[i] = tgamma(i + 1);
00489     }
00490     return factorials;
00491 }
00492
00495
00497 int MergedList::cmp_code_asc(const void *a, const void *b) {
00498     class MergedItem *x = *(class MergedItem **)a;
00499     class MergedItem *y = *(class MergedItem **)b;
00500
00501     return strcmp(x->get_code(), y->get_code());
00502 }
00503
00505 int MergedList::cmp_edges(const void *a, const void *b) {
00506     class MergedItemPair * x = * (class MergedItemPair **)a;
00507     class MergedItemPair * y = * (class MergedItemPair **)b;
00508
00509     return strcmp (x->get_item2()->get_code(), y->get_item2()->get_code());
00510 }
00511
00513 int MergedList::cmp_double(const void *a, const void *b) {
00514     double x = * (double *)a;
00515     double y = * (double *)b;
00516
00517     if (x > y) { return 1; }
00518     return -1;
00519 }
00520
00522 int MergedList::cmp_score_asc(const void *a, const void *b) {
00523     class MergedItem *x = *(class MergedItem **)a;
00524     class MergedItem *y = *(class MergedItem **)b;
00525
00526     if (x->get_final_score() == y->get_final_score()) {
```

```
00527         if (y->get_num_rankings() == x->get_num_rankings()) {
00528             return strcmp(x->get_code(), y->get_code());
00529         } else {
00530             return y->get_num_rankings() - x->get_num_rankings();
00531         }
00532     } else if (y->get_final_score() > x->get_final_score()) {
00533         return -1;
00534     } else {
00535         return 1;
00536     }
00537 }
00538
00540 int MergedList::cmp_score_desc(const void *a, const void *b) {
00541     class MergedItem *x = *(class MergedItem **)a;
00542     class MergedItem *y = *(class MergedItem **)b;
00543
00544     if (x->get_final_score() == y->get_final_score()) {
00545         if (y->get_num_rankings() == x->get_num_rankings()) {
00546             return strcmp(x->get_code(), y->get_code());
00547         } else {
00548             return y->get_num_rankings() - x->get_num_rankings();
00549         }
00550     } else if (y->get_final_score() > x->get_final_score()) {
00551         return 1;
00552     } else {
00553         return -1;
00554     }
00555 }
00556
00557 int MergedList::cmp_voter(const void *a, const void *b) {
00558     class Voter * x = *(class Voter **)a;
00559     class Voter * y = *(class Voter **)b;
00560
00561     if (x->get_weight() > y->get_weight()) { return 1; }
00562     return -1;
00563 }
00564
00566 rank_t MergedList::get_num_items() { return this->num_nodes; }
00567 class MergedItem * MergedList::get_item(uint32_t i) { return this->item_list[i]; }
00568 class MergedItem ** MergedList::get_item_list() { return this->item_list; }
00569 score_t MergedList::get_weight() { return this->weight; }
00570
00572 void MergedList::set_weight(score_t v) { this->weight = v; }
00573
00575 uint32_t MergedList::djb2(char * str) {
00576     unsigned long hash = 5381;
00577     int c;
00578
00579     while ((c = *str++))
00580         hash = ((hash << 5) + hash) + c; /* hash * 33 + c */
00581
00582     return hash;
00583 }
```

6.44 FLAGR/src/MergedList.h File Reference

Classes

- class MergedList

6.45 MergedList.h

Go to the documentation of this file.

```
00001 #ifndef MERGEDLIST_H
00002 #define MERGEDLIST_H
00003
00004
00005 class MergedList {
00006     private:
00007         uint32_t num_input_lists;
00008         class MergedItem ** hash_table;
00009         class MergedItem ** item_list;
00010         uint32_t mask;
00011         uint32_t num_slots;
```

```
00012         uint32_t num_nodes;
00013         uint32_t num_chains;
00014 
00015         score_t weight;
00016         double log10s[100000];
00017 
00018     private:
00019         uint32_t djb2(char *);
00020 
00021         static int cmp_code_asc(const void *, const void *);
00022         static int cmp_score_asc(const void *, const void *);
00023         static int cmp_score_desc(const void *, const void *);
00024         static int cmp_voter(const void *, const void *);
00025         static int cmp_edges(const void *, const void *);
00026         static int cmp_double(const void *, const void *);
00027 
00028         score_t * compute_state_matrix(class SimpleScoreStats *, class InputParams * );
00029         void matrixvec_multiply(score_t *, score_t *, score_t **);
00030 
00031         void permute(class MergedItem **, class InputList **, rank_t, score_t *, int l, int r);
00032 
00033         score_t stuart(double *, double *, double *, double *, char *);
00034         score_t sumStuart(double *, double, uint32_t, double *, double *);
00035         uint64_t factorial(uint32_t);
00036         double factorial(double);
00037         double * precompute_170_factorials();
00038         void compute_initial_weights(class InputList **);
00039 
00040     public:
00041         MergedList();
00042         MergedList(class InputList **, uint32_t, uint32_t);
00043         MergedList(uint32_t, uint32_t);
00044         ~MergedList();
00045 
00046         void insert(class InputItem *, uint32_t, class InputList **);
00047         void insert_merge(class MergedItem *, score_t);
00048         void convert_to_array();
00049         void display();
00050         void display_list();
00051         void write_to_CSV(char *, class InputParams *);
00052         void update_weight(char *, score_t);
00053         void reset_scores();
00054         void reset_weights();
00055         void rebuild(class InputList **);
00056         void clear_contents();
00057         void merge_with(class MergedList *, class InputParams *);
00058         rank_t get_item_rank(char *);
00059 
00061         void CombSUM(class InputList **, class SimpleScoreStats *, class InputParams *);
00062         void CombMNZ(class InputList **, class SimpleScoreStats *, class InputParams *);
00063         void CondorcetWinners(class InputList **, class SimpleScoreStats *, class InputParams *);
00064         void CopelandWinners(class InputList **, class SimpleScoreStats *, class InputParams *);
00065         void Outranking(class InputList **, class SimpleScoreStats *, class InputParams *);
00066         void KemenyOptimal(class InputList **, class SimpleScoreStats *, class InputParams *);
00067         void RobustRA(class InputList **, class SimpleScoreStats *, class InputParams *);
00068         void MC(class InputList **,  class SimpleScoreStats *, class InputParams *);
00069         class Voter ** DIBRA(class InputList **, class SimpleScoreStats *, class InputParams *);
00070         void PrefRel(class InputList **, class SimpleScoreStats *, class InputParams *);
00071         class MergedList * Agglomerative(class InputList **,  class SimpleScoreStats *, class
      InputParams *);
00072 
00074         void CustomMethod1(class InputList **,  class SimpleScoreStats *, class InputParams *);
00075         void CustomMethod2(class InputList **,  class SimpleScoreStats *, class InputParams *);
00076 
00078         double SpearmanRho(class InputList *);
00079         double SpearmanRho(class MergedList *);
00080         double ScaledFootruleDistance(class MergedList *);
00081         double ScaledFootruleDistance(uint32_t, class InputList *);
00082         double LocalScaledFootruleDistance(uint32_t, class InputList *);
00083         double CosineSimilarity(uint32_t, class InputList *);
00084         double KendallsTau(uint32_t, class InputList *);
00085 
00087         rank_t get_num_items();
00088         class MergedItem * get_item(uint32_t);
00089         class MergedItem ** get_item_list();
00090         score_t get_weight();
00091 
00093         void set_weight(score_t);
00094 };
00095 
00096 #endif // ITEMHASH_H
```

6.46 FLAGR/src/Query.cpp File Reference

```
#include "Query.h"
```

6.47 Query.cpp

Go to the documentation of this file.

```
00001 #include "Query.h"
00002
00005 Query::Query(uint32_t sw) :
00006         topic(NULL),
00007         agg(new Aggregator()),
00008         eval(NULL),
00009         real_experts_list(NULL),
00010         computed_experts_list(NULL) {
00011
00012             if (sw == 1) {
00013                 this->eval = new Evaluator();
00014             }
00015 }
00016
00018 Query::~Query() {
00019     if (this->real_experts_list) {
00020         for (uint32_t i = 0; i < this->agg->get_num_lists(); i++) {
00021             if (this->real_experts_list[i]) { delete this->real_experts_list[i]; }
00022         }
00023         delete [] this->real_experts_list;
00024         this->real_experts_list = NULL;
00025     }
00026
00027     if (this->computed_experts_list) {
00028         delete [] this->computed_experts_list;
00029         this->computed_experts_list = NULL;
00030     }
00031
00032     if (this->agg) {
00033         delete this->agg;
00034         this->agg = NULL;
00035     }
00036
00037     if (this->eval) {
00038         delete this->eval;
00039         this->eval = NULL;
00040     }
00041
00042     if (this->topic) {
00043         delete [] this->topic;
00044         this->topic = NULL;
00045     }
00046 }
00047
00049 class InputList * Query::create_list(char * v, double w) {
00050     return this->agg->create_list(v, w);
00051 }
00052
00054 void Query::aggregate(class InputParams * params) {
00055     this->computed_experts_list = this->agg->aggregate(this->topic, params);
00056 }
00057
00059 void Query::destroy_output_list() {
00060     if (this->computed_experts_list) {
00061         delete [] this->computed_experts_list;
00062     }
00063
00064     this->agg->destroy_output_list();
00065     this->eval->clear();
00066 }
00067
00069 void Query::insert_relev(char * v, uint32_t j) {
00070     this->eval->insert_relev(v, j);
00071 }
00072
00074 void Query::evaluate(rank_t ev_pts, FILE * e_file) {
00075     this->eval->evaluate(ev_pts, this->topic, this->agg->get_output_list(), e_file);
00076 }
00077
00078 double Query::evaluate_experts_list() {
```

```
00079     int32_t dis = 0;
00080     uint32_t num_lists = this->agg->get_num_lists();
00081     rank_t i = 0, j = 0;
00082     double spearmansRho = 0.0;
00083 
00084     if (this->real_experts_list && this->computed_experts_list) {
00085         for (i = 0; i < num_lists; i++) {
00086             class Voter * v1 = this->real_experts_list[i];
00087 
00088 //          printf("Voter %d:  %s - Real Rank:  %d (%5.3f), Computed Rank:
      ",i,v1->get_name(),i+1,v1->get_weight());
00089 //          printf("%d ", i+1);
00090 
00091             for (j = 0; j < num_lists; j++) {
00092                 class Voter * v2 = this->computed_experts_list[j];
00093                 if (strcmp(v1->get_name(), v2->get_name()) == 0) {
00094 //                  printf("%d (%5.3f)\n", j+1, v2->get_weight());
00095 //                  printf("%d a\n", j + 1);
00096                     dis += (i - j) * (i - j);
00097                     break;
00098                 }
00099             }
00100         }
00101 
00102         spearmansRho = 1.0 - (6.0 * dis) / (num_lists * (num_lists * num_lists - 1));
00103 //      printf("Spearman's Rho value = %5.3f\n", spearmansRho);
00104     }
00105     return spearmansRho;
00106 }
00107 
00109 void Query::evaluate_input() {
00110     double voter_map = 0.0;
00111     uint32_t num_lists = this->agg->get_num_lists(), i = 0;
00112     class InputList * inlist;
00113 
00114     this->real_experts_list = new Voter * [this->agg->get_num_lists()];
00115 
00116     for (i = 0; i < num_lists; i++) {
00117         inlist = this->agg->get_input_list(i);
00118 
00119         voter_map = this->eval->evaluate_input(inlist);
00120 
00121         this->real_experts_list[i] = new Voter(inlist->get_voter()->get_name(), voter_map);
00122     }
00123 
00124     qsort(this->real_experts_list, num_lists, sizeof(Voter *), &Query::cmp_voter);
00125 }
00126 
00128 void Query::init_weights() {
00129     this->agg->init_weights();
00130 }
00131 
00133 void Query::display() {
00134     printf("Displaying Data for Query:  %s\n", this->topic);
00135     this->agg->display();
00136 }
00137 
00139 void Query::display_relevs() {
00140     printf("Displaying Rels for Query:  %s\n", this->topic);
00141     this->eval->display_relevs();
00142 }
00143 
00145 class InputList * Query::get_input_list(uint32_t i) { return this->agg->get_input_list(i); }
00146 char * Query::get_topic() { return this->topic; }
00147 uint32_t Query::get_num_items() { return this->agg->get_num_items(); }
00148 rank_t Query::get_num_rel() { return this->eval->get_num_rel(); }
00149 rank_t Query::get_num_rel_ret() { return this->eval->get_true_positives(); }
00150 uint32_t Query::get_num_input_lists() { return this->agg->get_num_lists(); }
00151 double Query::get_average_precision() { return this->eval->get_average_precision(); }
00152 double Query::get_average_recall() { return this->eval->get_average_recall(); }
00153 double Query::get_average_dcg() { return this->eval->get_average_dcg(); }
00154 double Query::get_average_ndcg() { return this->eval->get_average_ndcg(); }
00155 double Query::get_precision(uint32_t i) { return this->eval->get_precision(i); }
00156 double Query::get_recall(uint32_t i) { return this->eval->get_recall(i); }
00157 double Query::get_F1(uint32_t i) { return this->eval->get_F1(i); }
00158 double Query::get_dcg(uint32_t i) { return this->eval->get_dcg(i); }
00159 double Query::get_ndcg(uint32_t i) { return this->eval->get_ndcg(i); }
00160 
00162 void Query::set_topic(char * v) {
00163     this->topic = new char[strlen(v) + 1];
00164     strcpy(this->topic, v);
00165 }
```

6.48 FLAGR/src/Query.h File Reference

Classes

- class Query

6.49 Query.h

Go to the documentation of this file.

```
00001 #ifndef QUERY_H
00002 #define QUERY_H
00003
00004
00005 class Query {
00006     private:
00007         char * topic;
00008         class Aggregator * agg;
00009         class Evaluator * eval;
00010
00011         class Voter ** real_experts_list;
00012         class Voter ** computed_experts_list;
00013
00014     private:
00015         static int cmp_voter(const void *a, const void *b) {
00016             class Voter * x = *(class Voter **)a;
00017             class Voter * y = *(class Voter **)b;
00018
00019             if (x->get_weight() > y->get_weight()) { return -1; }
00020             return 1;
00021         }
00022
00023     public:
00024         Query(uint32_t);
00025         ~Query();
00026
00027         class InputList * create_list(char *, double);
00028         void aggregate(class InputParams * params);
00029
00030         void insert_relev(char *, uint32_t);
00031         void display();
00032         void display_relevs();
00033         void evaluate(rank_t, FILE *);
00034         void evaluate_input();
00035         void destroy_output_list();
00036         double evaluate_experts_list();
00037         void init_weights();
00038
00040         uint32_t get_num_items();
00041         rank_t get_num_rel();
00042         rank_t get_num_rel_ret();
00043         uint32_t get_num_input_lists();
00044
00045         char * get_topic();
00046         double get_average_precision();
00047         double get_average_recall();
00048         double get_average_dcg();
00049         double get_average_ndcg();
00050         double get_precision(uint32_t);
00051         double get_recall(uint32_t);
00052         double get_F1(uint32_t);
00053         double get_dcg(uint32_t);
00054         double get_ndcg(uint32_t);
00055
00056         class InputList * get_input_list(uint32_t);
00057
00059         void set_topic(char *);
00060 };
00061
00062 #endif // QUERY_H
```

6.50 FLAGR/src/ram/Agglomerative.cpp File Reference

Classes

- struct max_similarity

Functions

- struct max_similarity * compute_similarities (struct max_similarity *max_similarities, class MergedList **templist, uint32_t nlists)

6.50.1 Function Documentation

6.50.1.1 compute_similarities()

```
struct max_similarity * compute_similarities (
            struct max_similarity * max_similarities,
            class MergedList ** templist,
            uint32_t nlists )
```

Determine the most similar MergedList of each MergedList. Instead on using a M x M matrix for all similarities, it suffices to use a M x 1 array which stores only the max similarity values. Compute all the pairwise list similarities, but store only the largest one for each list

Definition at line 152 of file Agglomerative.cpp.

6.51 Agglomerative.cpp

Go to the documentation of this file.

```
00001
00005
00008 struct max_similarity {
00009     score_t sim;
00010     int32_t merge_with;
00011 };
00012
00015 void MergedList::insert_merge(class MergedItem * item, score_t list_weight) {
00016     class MergedItem * q = NULL;
00017     class Ranking * rnk = NULL;
00018     score_t new_score = 0.0;
00019
00020     uint32_t HashValue = this->djb2(item->get_code()) & this->mask;
00021
00023     if (this->hash_table[HashValue] != NULL) {
00024
00026         for (q = this->hash_table[HashValue]; q != NULL; q = q->get_next()) {
00027
00029             if (strcmp(q->get_code(), item->get_code()) == 0) {
00030 //              printf("\t\tThe item was found!\n");
00031
00032                 for (rank_t itemr = 0; itemr < item->get_num_alloc_rankings(); itemr++) {
00033                     rnk = item->get_ranking(itemr);
00034
00035                     if (rnk->get_rank() != NOT_RANKED_ITEM_RANK) {
00036 //                      printf("\t\t\tThe item was ranked by list %d at rank %d\n",
      rnk->get_input_list()->get_id(), rnk->get_rank());
00037                         q->insert_ranking(rnk->get_input_list(), rnk->get_rank(), rnk->get_score());
00038 //                      printf("\t\t\tNew list node is\n"); q->display();
00039                     }
00040                 }
00041
00043                 new_score = ( this->weight * q->get_final_score() + list_weight *
      item->get_final_score() ) /
00044                         ( this->weight + list_weight );
00045 /*
00046 printf("New Score:  %5.3f --- (Old Score in List 1:  %5.3f, List 1 weight:  %5.3f,\
00047 Old Score in List 2:  %5.3f, List 2 weight:  %5.3f\n",
00048 new_score, q->get_final_score(), this->weight, item->get_final_score(), list_weight);
00049 */
00050                 q->set_final_score( new_score );
```

```
00051
00052                 return;
00053             }
00054         }
00055     }
00056
00058 //  printf("\t\tThe item was NOT found, creating a new node...\n");
00059     this->num_nodes++;
00060
00061     class MergedItem * record = new MergedItem(item);
00062
00064     record->set_next(this->hash_table[HashValue]);
00065     this->hash_table[HashValue] = record;
00066 }
00067
00068
00070 void MergedList::merge_with(class MergedList * inlist, class InputParams * params) {
00071     rank_t r = 0;
00072     score_t c1 = params->get_c1(), c2 = params->get_c2();
00073
00075     if (this->item_list) {
00076         delete [] this->item_list;
00077     }
00078
00080     for (r = 0; r < inlist->get_num_items(); r++) {
00081 //      printf("\tSearching for the item %d (%s) of the second list in the first list...\n", r,
      in->get_item(r)->get_code());
00082         this->insert_merge( inlist->get_item(r), inlist->get_weight() );
00083 //      getchar();
00084     }
00085
00086 //  printf("Old weight 1:  %5.3f - Old weight 2:  %5.3f ", this->weight, inlist->get_weight());
00087
00089     this->weight = (c1 * this->weight + c2 * inlist->get_weight()) / (c1 + c2);
00090
00091 //  printf("New weight:  %5.3f\n", this->weight); getchar();
00092
00094     this->convert_to_array();
00095     qsort(this->item_list, this->num_nodes, sizeof(class MergedItem *), &MergedList::cmp_score_desc);
00096 }
00097
00098
00100 double MergedList::SpearmanRho(class MergedList * inlist) {
00101     class MergedItem * q, * p;
00102     double rho = 0.0, sum = 0.0;
00103     rank_t n1 = this->num_nodes, n2 = inlist->get_num_items();
00104
00105     double denom = pow(n1, 3) - n1;
00106
00107     for (rank_t i = 0; i < n1; i++) {
00108         q = this->item_list[i];
00109
00110         for (rank_t j = 0; j < n2; j++) {
00111             p = inlist->get_item(j);
00112 //          printf("Comparing (%s, %d) with (%s, %d)\n", q->get_code(), i, p->get_code(), j);
00113
00114             if (strcmp(q->get_code(), p->get_code()) == 0) {
00115                 sum += (i - j) * (i - j);
00116                 break;
00117             }
00118         }
00119     }
00120
00121     rho = 1.0 - 6.0 * sum / denom;
00122 //  printf("(%d-%d) - rho=%5.3f - sum:%5.3f\n", this->num_nodes, in->get_num_items(), rho, sum);
00123
00124     return rho;
00125 }
00126
00128 double MergedList::ScaledFootruleDistance(class MergedList * inlist) {
00129     double d = 0.0, nd = 0.0;
00130     rank_t i = 0, r = 0, R = inlist->get_num_items(), L = this->num_nodes;
00131
00132     for (i = 0; i < L; i++) {
00133         for (r = 0; r < R; r++) {
00134             if (strcmp(this->item_list[i]->get_code(), inlist->get_item(r)->get_code()) == 0) {
00135                 d += fabs( (double) i / L - (double) r / R );
00136                 break;
00137             }
00138         }
00139     }
00140
00141     nd = 2.00 * d / R;
00142
00143 //  if (nd > 1) { printf("gt > 1:  ==%d== d=%5.3f NormD=%5.3f", R, d, nd); getchar(); }
00144 //  printf("SFD: %5.3f (Norm SFD: %5.3f) - Items:  %d\n", d, nd, R); getchar();
00145
```

```
00146     return nd;
00147
00148 }
00149
00152 struct max_similarity * compute_similarities(struct max_similarity * max_similarities,
00153     class MergedList ** templist, uint32_t nlists) {
00154
00155         uint32_t m = 0, n = 0;
00156         score_t sim = 0.0, max_sim = -2.0;
00157
00159         for (m = 0; m < nlists; m++) {
00160             max_similarities[m].sim = -2.0;
00161             max_similarities[m].merge_with = -1;
00162         }
00163
00164         for (m = 0; m < nlists; m++) {
00165 //          if (templist[m]) { templist[m]->display_list(); getchar(); }
00166             max_sim = -2.0;
00167             for (n = 0; n < nlists; n++) {
00168                 if (m != n && templist[m] && templist[n]) {
00169                     sim = templist[m]->SpearmanRho(templist[n]);
00170
00171 //                  printf("\nRho correlation between %d and %d = %5.3f\n", m, n, sim);
00172                     if (sim > max_sim) {
00173                         max_sim = sim;
00174                         max_similarities[m].sim = max_sim;
00175                         max_similarities[m].merge_with = n;
00176                     }
00177                 }
00178             }
00179         }
00180         return max_similarities;
00181 }
00182
00184 void MergedList::compute_initial_weights(class InputList ** input_lists) {
00185     rank_t k = 0, p_rank = 0, q_rank = 0, i = 0, j = 0;
00186     uint32_t nlists = this->num_input_lists, l = 0;
00187     score_t wins = 0, losses = 0;
00188
00189     class Voter * v = NULL;
00190     class MergedItem *p = NULL, *q = NULL;
00191     class MergedList * temp_agglist = new MergedList(1024, nlists);
00192
00193     for (l = 0; l < nlists; l++) {
00194         for (k = 0; k < input_lists[l]->get_num_items(); k++) {
00195             temp_agglist->insert(input_lists[l]->get_item(k), l, input_lists);
00196         }
00197     }
00198     temp_agglist->convert_to_array();
00199
00200     for (i = 0; i < temp_agglist->get_num_items(); i++) {
00201         p = temp_agglist->get_item(i);
00202
00203         for (j = i + 1; j < temp_agglist->get_num_items(); j++) {
00204             q = temp_agglist->get_item(j);
00205
00206             wins = 0; losses = 0;
00207
00208             for (k = 0; k < nlists; k++) {
00209                 p_rank = p->get_ranking(k)->get_rank();
00210                 q_rank = q->get_ranking(k)->get_rank();
00211
00212                 if (p_rank < q_rank) {
00213                     wins++;
00214                 } else if (p_rank > q_rank) {
00215                     losses++;
00216                 }
00217
00218 //              printf("\nLIST %d:  Item %d (%s) rank %d (VS) item %d (%s) rank %d\n",
00219 //                  k, i, p->get_code(), p_rank, j, q->get_code(), q_rank);
00220             }
00221
00224 //          printf("Item %d (%s) -VS- item %d (%s) -- W: %5.3f - L: %5.3f\n", i, p->get_code(),
00225 //              j, q->get_code(), wins, losses);
00226
00227             for (k = 0; k < nlists; k++) {
00228                 p_rank = p->get_ranking(k)->get_rank();
00229                 q_rank = q->get_ranking(k)->get_rank();
00230
00231                 if (wins > losses && p_rank < q_rank) {
00232                     v = p->get_ranking(k)->get_input_list()->get_voter();
00233                     v->set_weight(v->get_weight() + 1);
00234 //                  printf("%s (LIST %d) Agrees:  - New Weight:  %5.3f\n", v->get_name(), k,
    v->get_weight());
00235                 } else if (wins < losses && p_rank > q_rank) {
00236                     v = q->get_ranking(k)->get_input_list()->get_voter();
00237                     v->set_weight(v->get_weight() + 1);
```

```
00238 //                  printf("%s (LIST %d) Agrees:  - New Weight:  %5.3f\n", v->get_name(), k,
      v->get_weight());
00239                   }
00240               }
00241 //            getchar();
00242           }
00243       }
00244
00245       delete temp_agglist;
00246 }
00247
00248
00249 class MergedList * MergedList::Agglomerative(class InputList ** inlists, class SimpleScoreStats * s,
      class InputParams * prms) {
00250       uint32_t m = 0, next = 0;
00251       score_t max_sim = -2.0;
00252       uint32_t nlists = this->num_input_lists, remaining_lists = this->num_input_lists, msl = 0;
00253
00255       struct max_similarity * max_similarities = new max_similarity [nlists];
00256
00257       this->compute_initial_weights(inlists);
00258
00260       class MergedList ** templist = new MergedList * [nlists];
00261
00262       for (m = 0; m < nlists; m++) {
00263           templist[m] = new MergedList(inlists, nlists, m);
00264 //        templist[m]->display(); getchar();
00265       }
00266
00268       max_similarities = compute_similarities(max_similarities, templist, nlists);
00269
00271       while(remaining_lists > 1) {
00272
00274           max_sim = -2.0;
00275           for (m = 0; m < nlists; m++) {
00276 //            printf("Most similar list to %d is list %d:  %5.3f\n", m, max_similarities[m].merge_with,
      max_similarities[m].sim);
00277
00278               if (max_similarities[m].sim > max_sim) {
00279                   max_sim = max_similarities[m].sim;
00280                   next = m;
00281               }
00282           }
00283
00285           msl = max_similarities[next].merge_with;
00286 //        printf("Next merge:  list %d with list %d\n", next, msl );
00287
00288           templist[next]->merge_with( templist[ msl ], prms );
00289
00291           delete templist[ msl ];
00292           templist[ msl ] = NULL;
00293
00295           max_similarities = compute_similarities(max_similarities, templist, nlists);
00296
00298           remaining_lists--;
00299
00300 //        printf("Merge completed, Remaining lists = %d\n\n", remaining_lists );
00301 //        templist[next]->display();
00302       }
00303
00305 //  delete templist[next];
00306       class MergedList * ret = templist[next];
00307       delete [] templist;
00308       delete [] max_similarities;
00309
00310       return ret;
00311 }
```

6.52 FLAGR/src/ram/CombMNZ.cpp File Reference

6.53 CombMNZ.cpp

Go to the documentation of this file.

```
00001
00006
00007 void MergedList::CombMNZ(class InputList ** inlists, class SimpleScoreStats * s, class InputParams *
      prms) {
00008       class MergedItem * q;
```

```
00009     class Ranking * r;
00010     class InputList * l;
00011     double score = 0.0, voter_weight = 1.0;
00012 
00013     uint32_t weights_norm = prms->get_weights_normalization();
00014     uint32_t ram = prms->get_aggregation_method();
00015     uint32_t rank_hits = 0;
00016 
00018     for (rank_t i = 0; i < this->num_nodes; i++) {
00019         q = this->item_list[i];
00020         rank_hits = q->get_num_rankings();
00021 
00022         for (uint32_t j = 0; j < q->get_num_alloc_rankings(); j++) {
00023             r = q->get_ranking(j);
00024             l = r->get_input_list();
00025 
00027             voter_weight = l->get_voter()->get_weight();
00028 
00030             if (weights_norm == 2) {
00031                 voter_weight = (voter_weight - s->get_min_val()) / (s->get_max_val() -
      s->get_min_val());
00032 
00034             } else if (weights_norm == 3) {
00035                 voter_weight = voter_weight * s->get_std_val() * s->get_std_val() / s->get_max_val();
00036 
00038             } else if (weights_norm == 4) {
00039                 voter_weight = voter_weight / s->get_max_val();
00040             }
00041 
00043             if(l && r->get_rank() != NOT_RANKED_ITEM_RANK) {
00044 //              printf("Real Voter weight:  %5.3f - Normalized:  %5.3f\n",
00045 //                          l->get_voter()->get_weight(), voter_weight);
00046 
00048                 if(ram == 110 || ram == 5110) {
00049                     score = (this->num_nodes - r->get_rank() + 1.0) / (score_t)this->num_nodes;
00050 
00052                 } else if (ram == 111 || ram == 5111) {
00053                     score = (l->get_num_items() - r->get_rank() + 1.0) / (score_t)l->get_num_items();
00054 
00056                 } else if (ram == 112 || ram == 5112) {
00057                     score = (r->get_score() - l->get_min_score()) / (l->get_max_score() -
      l->get_min_score());
00058 
00060                 } else if (ram == 113 || ram == 5113) {
00061                     score = (r->get_score() - l->get_mean_score()) / l->get_std_score();
00062 
00064                 } else if (ram == 114 || ram == 5114) {
00065                     score = (this->num_nodes - r->get_rank() + 1.0) / (score_t)this->num_nodes;
00066                 }
00067 
00068 
00070             } else {
00072                 if(ram == 110 || ram == 5110) {
00073                     score = (this->num_nodes - l->get_num_items() + 1.0) / (2.0 * this->num_nodes);
00074 
00077                 } else if(ram == 111 || ram == 112 || ram == 113 || ram == 114 ||
00078                     ram == 5111 || ram == 5112 || ram == 5113 || ram == 5114) {
00079                         score = 0.0;
00080                 }
00081             }
00082 
00083             score *= rank_hits * voter_weight;
00084 
00085             q->set_final_score(q->get_final_score() + score);
00086         }
00087     }
00088 
00089     qsort(this->item_list, this->num_nodes, sizeof(class MergedItem *), &MergedList::cmp_score_desc);
00090 
00091 //  this->display_list(); getchar();
00092 }
```

6.54 FLAGR/src/ram/CombSUM.cpp File Reference

6.55 CombSUM.cpp

Go to the documentation of this file.

```
00001
```

```
00006
00007 void MergedList::CombSUM(class InputList ** inlists, class SimpleScoreStats * s, class InputParams *
      prms) {
00008     class MergedItem * q;
00009     class Ranking * r;
00010     class InputList * l;
00011     double score = 0.0, voter_weight = 1.0;
00012
00013     uint32_t weights_norm = prms->get_weights_normalization();
00014     uint32_t ram = prms->get_aggregation_method();
00015
00017     for (rank_t i = 0; i < this->num_nodes; i++) {
00018         q = this->item_list[i];
00019
00020         for (uint32_t j = 0; j < q->get_num_alloc_rankings(); j++) {
00021             r = q->get_ranking(j);
00022             l = r->get_input_list();
00023
00025             voter_weight = l->get_voter()->get_weight();
00026
00028             if (weights_norm == 2) {
00029                 voter_weight = (voter_weight - s->get_min_val()) / (s->get_max_val() -
      s->get_min_val());
00030
00032             } else if (weights_norm == 3) {
00033                 voter_weight = voter_weight * s->get_std_val() * s->get_std_val() / s->get_max_val();
00034
00036             } else if (weights_norm == 4) {
00037                 voter_weight = voter_weight / s->get_max_val();
00038             }
00039
00041             if(l && r->get_rank() != NOT_RANKED_ITEM_RANK) {
00042 //              printf("Real Voter weight:  %5.3f - Normalized:  %5.3f\n",
00043 //                          l->get_voter()->get_weight(), voter_weight);
00044
00046                 if(ram == 100 || ram == 5100) {
00047                     score = (this->num_nodes - r->get_rank() + 1.0) / (score_t)this->num_nodes;
00048
00050                 } else if (ram == 101 || ram == 5101) {
00051                     score = (l->get_num_items() - r->get_rank() + 1.0) / (score_t)l->get_num_items();
00052
00054                 } else if (ram == 102 || ram == 5102) {
00055                     score = (r->get_score() - l->get_min_score()) / (l->get_max_score() -
      l->get_min_score());
00056
00058                 } else if (ram == 103 || ram == 5103) {
00059                     score = (r->get_score() - l->get_mean_score()) / l->get_std_score();
00060
00062                 } else if (ram == 104 || ram == 5104) {
00063                     score = (this->num_nodes - r->get_rank() + 1.0) / (score_t)this->num_nodes;
00064                 }
00065
00067             } else {
00069                 if(ram == 100 || ram == 5100) {
00070                     score = (this->num_nodes - l->get_num_items() + 1.0) / (2.0 * this->num_nodes);
00071
00074                 } else if(ram == 101 || ram == 102 || ram == 103 || ram == 104 ||
00075                     ram == 5101 || ram == 5102 || ram == 5103 || ram == 5104) {
00076                         score = 0.0;
00077                 }
00078             }
00079
00080             score *= voter_weight;
00081
00082             q->set_final_score(q->get_final_score() + score);
00083         }
00084 //      printf("%d :  Item Score:  %5.3f\n", i, q->get_score());
00085     }
00086
00087     qsort(this->item_list, this->num_nodes, sizeof(class MergedItem *), &MergedList::cmp_score_desc);
00088
00089 //  this->display_list(); getchar();
00090 }
```

6.56 FLAGR/src/ram/CondorcetWinners.cpp File Reference

6.57 CondorcetWinners.cpp

Go to the documentation of this file.

```
00001
00003
00004 void MergedList::CondorcetWinners(class InputList ** inlists, class SimpleScoreStats * s, class
      InputParams * prms) {
00005
00006     class MergedItem *p, *q;
00007     bool verbose = false;
00008     rank_t p_rank = 0, q_rank = 0;
00009     score_t wins = 0.0, losses = 0.0, ties = 0.0, voter_weight = 1.0;
00010
00011     uint32_t weights_norm = prms->get_weights_normalization();
00012
00013     for (rank_t i = 0; i < this->num_nodes; i++) {
00014         p = this->item_list[i];
00015
00016         for (rank_t j = i + 1; j < this->num_nodes; j++) {
00017             q = this->item_list[j];
00018
00019             wins = 0; losses = 0; ties = 0;
00020
00021             for (uint32_t k = 0; k < this->num_input_lists; k++) {
00022                 voter_weight = p->get_ranking(k)->get_input_list()->get_voter()->get_weight();
00023
00025                 if (weights_norm == 2) {
00026                     voter_weight = (voter_weight - s->get_min_val()) / (s->get_max_val() -
      s->get_min_val());
00027
00029                 } else if (weights_norm == 3) {
00030                     voter_weight = voter_weight * s->get_std_val() * s->get_std_val() /
      s->get_max_val();
00031
00033                 } else if (weights_norm == 4) {
00034                     voter_weight = voter_weight / s->get_max_val();
00035                 }
00036
00037                 p_rank = p->get_ranking(k)->get_rank();
00038                 q_rank = q->get_ranking(k)->get_rank();
00039
00040 //                printf("%d:  %5.3f-%5.3f - (%d,%d)\n",
00041 //                    k, p->get_ranking(k)->get_input_list()->get_voter()->get_weight(),
00042 //                    q->get_ranking(k)->get_input_list()->get_voter()->get_weight(), p_rank, q_rank);
00043
00044                 if (p_rank < q_rank) {
00045                     wins += voter_weight;
00046                     if (verbose) { printf("WIN %5.3f:  ", wins); }
00047                 } else if (p_rank == q_rank) {
00048                     ties += voter_weight;
00049                     if (verbose) { printf("TIE %5.3f:  ", ties); }
00050                 } else {
00051                     losses += voter_weight;
00052                     if (verbose) { printf("LOSS %5.3f:  ", losses); }
00053                 }
00054
00055                 if (verbose) {
00056                     printf("\nLIST %d:  Item %d (%s) rank %d (VS) item %d (%s) rank %d\n",
00057                         k, i, p->get_code(), p_rank, j, q->get_code(), q_rank);
00058                 }
00059             }
00060
00061             if (wins > losses) {
00062                 if (verbose) {
00063                     printf("Item %d (%s) WINS item %d (%s) -- %5.3f-%5.3f-%5.3f\n", i, p->get_code(),
00064                         j, q->get_code(), wins, ties, losses); getchar();
00065                 }
00066
00067                 p->set_final_score(p->get_final_score() + 1);
00068
00069             } else if (wins < losses) {
00070                 if (verbose) {
00071                     printf("Item %d (%s) LOSSES FROM item %d (%s) -- %5.3f-%5.3f-%5.3f\n",
00072                         i, p->get_code(), j, q->get_code(), wins, ties, losses);
00073                     getchar();
00074                 }
00075                 q->set_final_score(q->get_final_score() + 1);
00076             }
00077         }
00078     }
00079
00080     qsort(this->item_list, this->num_nodes, sizeof(class MergedItem *), &MergedList::cmp_score_desc);
00081 }
```

6.58 FLAGR/src/ram/CopelandWinners.cpp File Reference

6.59 CopelandWinners.cpp

Go to the documentation of this file.

```
00001
00003
00004 void MergedList::CopelandWinners(class InputList ** inlists, class SimpleScoreStats * s, class
      InputParams * prms) {
00005
00006     class MergedItem *p, *q;
00007     bool verbose = false;
00008     rank_t p_rank = 0, q_rank = 0;
00009     score_t wins = 0.0, losses = 0.0, ties = 0.0, voter_weight = 1.0;
00010
00011     uint32_t weights_norm = prms->get_weights_normalization();
00012
00013     for (rank_t i = 0; i < this->num_nodes; i++) {
00014         p = this->item_list[i];
00015
00016         for (rank_t j = i + 1; j < this->num_nodes; j++) {
00017             q = this->item_list[j];
00018
00019             wins = 0; losses = 0; ties = 0;
00020
00021             for (uint32_t k = 0; k < this->num_input_lists; k++) {
00022                 voter_weight = p->get_ranking(k)->get_input_list()->get_voter()->get_weight();
00023
00025                 if (weights_norm == 2) {
00026                     voter_weight = (voter_weight - s->get_min_val()) / (s->get_max_val() -
      s->get_min_val());
00027
00029                 } else if (weights_norm == 3) {
00030                     voter_weight = voter_weight * s->get_std_val() * s->get_std_val() /
      s->get_max_val();
00031
00033                 } else if (weights_norm == 4) {
00034                     voter_weight = voter_weight / s->get_max_val();
00035                 }
00036
00037                 p_rank = p->get_ranking(k)->get_rank();
00038                 q_rank = q->get_ranking(k)->get_rank();
00039
00040 //                printf("%d:  %5.3f-%5.3f - (%d,%d)\n",
00041 //                    k, p->get_ranking(k)->get_input_list()->get_voter()->get_weight(),
00042 //                    q->get_ranking(k)->get_input_list()->get_voter()->get_weight(), p_rank, q_rank);
00043
00044                 if (p_rank < q_rank) {
00045                     wins += voter_weight;
00046                     if (verbose) { printf("WIN %5.3f:  ", wins); }
00047                 } else if (p_rank == q_rank) {
00048                     ties += voter_weight;
00049                     if (verbose) { printf("TIE %5.3f:  ", ties); }
00050                 } else {
00051                     losses += voter_weight;
00052                     if (verbose) { printf("LOSS %5.3f:  ", losses); }
00053                 }
00054
00055                 if (verbose) {
00056                     printf("\nLIST %d:  Item %d (%s) rank %d (VS) item %d (%s) rank %d\n",
00057                         k, i, p->get_code(), p_rank, j, q->get_code(), q_rank);
00058                 }
00059             }
00060
00061             if (wins > losses) {
00062                 if (verbose) {
00063                     printf("Item %d (%s) WINS item %d (%s) -- %5.3f-%5.3f-%5.3f\n", i, p->get_code(),
00064                         j, q->get_code(), wins, ties, losses); getchar();
00065                 }
00066
00067                 p->set_final_score(p->get_final_score() + 1);
00068
00069             } else if (wins == losses) {
00070                 if (verbose) {
00071                     printf("Item %d (%s) IS IN TIE item %d (%s) -- %5.3f-%5.3f-%5.3f\n",
00072                         i, p->get_code(), j, q->get_code(), wins, ties, losses);
00073                     getchar();
00074                 }
00075                 p->set_final_score(p->get_final_score() + 0.5);
00076                 q->set_final_score(q->get_final_score() + 0.5);
00077
00078             } else if (wins < losses) {
```

```
00079                 if (verbose) {
00080                     printf("Item %d (%s) LOSSES FROM item %d (%s) -- %5.3f-%5.3f-%5.3f\n",
00081                         i, p->get_code(), j, q->get_code(), wins, ties, losses);
00082                     getchar();
00083                 }
00084                 q->set_final_score(q->get_final_score() + 1);
00085             }
00086         }
00087     }
00088
00089     qsort(this->item_list, this->num_nodes, sizeof(class MergedItem *), &MergedList::cmp_score_desc);
00090 }
```

6.60 FLAGR/src/ram/CustomMethods.cpp File Reference

6.61 CustomMethods.cpp

Go to the documentation of this file.

```
00001
00004
00006 void MergedList::CustomMethod1(class InputList ** inlists, class SimpleScoreStats * s, class
      InputParams * prms) {
00008     class MergedItem * q;
00009     class Ranking * r;
00010
00012     for (rank_t i = 0; i < this->num_nodes; i++) {
00013         q = this->item_list[i];
00014
00016         for (uint32_t j = 0; j < q->get_num_alloc_rankings(); j++) {
00017             r = q->get_ranking(j);
00018             r->display();
00019         }
00020     }
00021
00023     qsort(this->item_list, this->num_nodes, sizeof(class MergedItem *), &MergedList::cmp_score_desc);
00024
00027 }
00028
00029
00031 void MergedList::CustomMethod2(class InputList ** inlists, class SimpleScoreStats * s, class
      InputParams * prms) {
00033
00035     qsort(this->item_list, this->num_nodes, sizeof(class MergedItem *), &MergedList::cmp_score_desc);
00036
00039 }
```

6.62 FLAGR/src/ram/DIBRA.cpp File Reference

6.63 DIBRA.cpp

Go to the documentation of this file.

```
00001
00007
00008 class Voter ** MergedList::DIBRA(class InputList ** inlists, class SimpleScoreStats * s, class
      InputParams * prms) {
00009
00010     double prev_weight = 0.0;
00011     score_t min_d = 1000.0, max_d = 0.0, sum_w = 0.0, mean_w = 0.0;
00012     score_t w[this->num_input_lists], dis[this->num_input_lists];
00013
00014     uint32_t z = 0, cutoff = 0, max_cutoff = 0;
00015     uint32_t ram = prms->get_aggregation_method();
00016     uint32_t corel = prms->get_correlation_method();
00017     int32_t iteration = 0;
00018     class Voter ** voters_list = new Voter * [this->num_input_lists];
00019
00021     for (z = 0; z < this->num_input_lists; z++) {
00022         inlists[z]->set_voter_weight( 1.0 );
00023     }
```

```
00024
00025     uint32_t conv = 1;
00026
00028     while (1) {
00030         this->reset_scores();
00031
00033         if (ram >= 5100 && ram <= 5109) {
00034             this->CombSUM(inlists, s, prms);
00035
00036         } else if (ram >= 5110 && ram <= 5119) {
00037             this->CombMNZ(inlists, s, prms);
00038
00039         } else if (ram == 5200) {
00040             this->CondorcetWinners(inlists, s, prms);
00041
00042         } else if (ram == 5201) {
00043             this->CopelandWinners(inlists, s, prms);
00044
00045         } else if (ram == 5300) {
00046             this->Outranking(inlists, s, prms);
00047
00048         }
00049
00051         s->set_min_val(1000.0); s->set_max_val(0.0); s->set_mean_val(0.0); s->set_std_val(0.0);
00052         min_d = 1000.0; max_d = 0.0; sum_w = 0.0;
00053
00055         for (z = 0; z < this->num_input_lists; z++) {
00056             if (corel == 1) {
00057                 dis[z] = this->SpearmanRho(inlists[z]);
00058
00059             } else if (corel == 2) {
00060                 dis[z] = this->ScaledFootruleDistance(z, inlists[z]);
00061
00062             } else if (corel == 3) {
00063                 dis[z] = this->CosineSimilarity(z, inlists[z]);
00064
00065             } else if (corel == 4) {
00066                 dis[z] = this->LocalScaledFootruleDistance(z, inlists[z]);
00067
00068             } else if (corel == 5) {
00069                 dis[z] = this->KendallsTau(z, inlists[z]);
00070             }
00071
00074             if (dis[z] > max_d) { max_d = dis[z]; }
00075             if (dis[z] < min_d) { min_d = dis[z]; }
00076             if (inlists[z]->get_cutoff() > max_cutoff) {
00077                 max_cutoff = inlists[z]->get_cutoff();
00078             }
00079         }
00080
00081         conv = 1;
00082
00084         for (z = 0; z < this->num_input_lists; z++) {
00085
00086             prev_weight = inlists[z]->get_voter()->get_weight();
00087
00089             if (dis[z] > 0) {
00090 //              if (z == 1) { w[z] = 1.0; } else { w[z] = 0.0; }
00091 //              w[z] = prev_weight + 1.0 / ( 1.0 + exp( params->get_gamma() * (iteration + 1.00) *
    dis[z] )); /// Logistic
00092                 w[z] = prev_weight + exp(- prms->get_gamma() * ( iteration + 1.00) * dis[z] );
00093 //              w[z] = prev_weight + (score_t)1.0 / (params->get_gamma() * (iteration + 1.0) *
    (iteration + 1.0) * tanh( dis[z] )); /// tanh
00094             } else {
00095                 w[z] = prev_weight;
00096             }
00097
00098 //          printf("Iteration:  %d.  Voter %d (%s) distance:  %5.3f - weight:  %5.3f (prev:  %5.3f -
    diff:  %5.3f) , %5.3f\n",
00099 //              iteration, z, inlists[z]->get_voter()->get_name(), dis[z], w[z],
00100 //              prev_weight, w[z] - prev_weight, tanh( dis[z] ));
00101
00103             if (w[z] > s->get_max_val()) { s->set_max_val(w[z]); }
00104             if (w[z] < s->get_min_val()) { s->set_min_val(w[z]); }
00105             sum_w += w[z];
00106
00107             if (w[z] - prev_weight > prms->get_convergence_precision()) {
00108                 conv = 0;
00109             }
00110         }
00111
00112         mean_w = sum_w / (score_t) this->num_input_lists;
00113         s->set_mean_val(mean_w);
00114         sum_w = 0.0;
00115         for (z = 0; z < this->num_input_lists; z++) {
00116             sum_w += (w[z] - mean_w) * (w[z] - mean_w);
00117         }
```

```
00118         s->set_std_val( sqrt(sum_w / (double)this->num_input_lists) );
00119
00121         for (z = 0; z < this->num_input_lists; z++) {
00122 //          printf("Voter %d (%s) distance:  %5.3f - weight:  %5.3f\n", z,
      inlists[z]->get_voter()->get_name(), dis[z], w[z]);
00123             inlists[z]->set_voter_weight( w[z] );
00124
00125             voters_list[z] = inlists[z]->get_voter();
00126         }
00127 //      getchar();
00128
00129         qsort(voters_list, this->num_input_lists, sizeof(Voter *), &MergedList::cmp_voter);
00130         iteration++;
00131
00132         if (conv == 1 || iteration > prms->get_max_iterations()) {
00133             break;
00134         }
00135     }
00136
00137 /*
00138 printf("\n");
00139 for (z = 0; z < this->num_input_lists; z++) {
00140 printf("%s|%5.3f\n", inlists[z]->get_voter()->get_name(), inlists[z]->get_voter()->get_weight());
00141 }
00142 getchar();
00143 */
00144
00148     if (prms->get_list_pruning()) {
00149         for (z = 0; z < this->num_input_lists; z++) {
00150
00151             score_t nw = (w[z] - s->get_min_val()) / (s->get_max_val() - s->get_min_val());
00152
00153 //          cutoff = (double)this->num_nodes / (double)inlists[z]->get_num_items() +
00154 //              nw * this->num_input_lists * log10(10.0 + (double)inlists[z]->get_num_items());
00155
00156             cutoff = (prms->get_delta1() + prms->get_delta2() * nw) * inlists[z]->get_num_items();
00157
00158 //          printf("Cutoff for voter %d (%5.3f - %s):  %d\n", z, nw,
      inlists[z]->get_voter()->get_name(), cutoff); getchar();
00159
00160             if (cutoff >= inlists[z]->get_num_items()) {
00161                 inlists[z]->set_cutoff(inlists[z]->get_num_items());
00162             } else {
00163                 inlists[z]->set_cutoff(cutoff);
00164             }
00165         }
00166
00167         this->rebuild(inlists);
00168
00169         if (ram >= 5100 && ram <= 5109) {
00170             this->CombSUM(inlists, s, prms);
00171
00172         } else if (ram >= 5110 && ram <= 5119) {
00173             this->CombMNZ(inlists, s, prms);
00174
00175         } else if (ram == 5200) {
00176             this->CondorcetWinners(inlists, s, prms);
00177
00178         } else if (ram == 5201) {
00179             this->CopelandWinners(inlists, s, prms);
00180
00181         } else if (ram == 5300) {
00182             this->Outranking(inlists, s, prms);
00183         }
00184     }
00185
00186 //  printf(" Nodes:  %d", this->num_nodes);
00187     return voters_list;
00188 }
```

6.64 FLAGR/src/ram/KemenyOptimal.cpp File Reference

Functions

- void swap (class MergedItem **x, class MergedItem **y)

6.64.1 Function Documentation

6.64.1.1 swap()

```
void swap (
            class MergedItem ** x,
            class MergedItem ** y )
```

The Brute Force approach for Kemeny Optimal Aggregation. It constructs all possible permutations and returns the one that minimizes (maximizes) the distance (correlation) from (with) all input preference lists. Note: Combinatorial complexity - Do not try that with aggregate lists with more than 10 items!

Definition at line 7 of file KemenyOptimal.cpp.

6.65 KemenyOptimal.cpp

Go to the documentation of this file.

```
00001
00005
00006 /* Function to swap values at two pointers */
00007 void swap(class MergedItem **x, class MergedItem **y) {
00008     class MergedItem * temp;
00009     temp = *x;
00010     *x = *y;
00011     *y = temp;
00012 }
00013
00014 void MergedList::permute(class MergedItem ** best, class InputList ** inlists, rank_t n_items, score_t
      * max_cor, int l, int r)  {
00015     uint32_t j = 0;
00016     rank_t rnk = 0;
00017     score_t cor = 0.0;
00018
00020     if (l == r) {
00021         for (rnk = 0; rnk < n_items; rnk++) {
00022             this->item_list[rnk]->set_final_ranking(rnk + 1);
00023         }
00024
00025         for (j = 0; j < this->num_input_lists; j++) {
00026             cor += this->KendallsTau( 0, inlists[j] );
00027 //          printf("Cumulative correlation of this permutation with list %d:  %5.3f\n", i, cor);
00028         }
00029 //      printf("Correlation of this permutation with all lists:  %5.3f\n", cor);
00030
00031         if (cor > *max_cor) {
00032             *max_cor = cor;
00033             for (rnk = 0; rnk < n_items; rnk++) {
00034                 best[rnk] = this->item_list[rnk];
00035             }
00036
00037 //          for (rnk = 0; rnk < n_items; rnk++) {
00038 //              printf("best list item %d:  %s\n", rnk, best[rnk]->get_code());
00039 //          }
00040         }
00041     } else {
00043         for (int i = l; i <= r; i++) {
00044
00046             swap(&(this->item_list[l]), &(this->item_list[i]));
00047
00049             this->permute(best, inlists, n_items, max_cor, l + 1, r);
00050
00052             swap(&(this->item_list[l]), &(this->item_list[i]));
00053         }
00054     }
00055 }
00056
00057 void MergedList::KemenyOptimal(class InputList ** inlists, class SimpleScoreStats * s, class
      InputParams * prms) {
00058     rank_t i = 0;
00059     score_t max_cor = 0;
00060
00061     class MergedItem ** best_list = new MergedItem * [this->num_nodes];
00062     for (i = 0; i < this->num_nodes; i++) {
00063         best_list[i] = this->item_list[i];
00064     }
00065
00066     permute(best_list, inlists, this->num_nodes, &max_cor, 0, this->num_nodes - 1);
```

```
00067
00068     for (rank_t rnk = 0; rnk < this->num_nodes; rnk++) {
00069         this->item_list[rnk] = best_list[rnk];
00070         this->item_list[rnk]->set_final_ranking(rnk + 1);
00071         this->item_list[rnk]->set_final_score(this->num_nodes - rnk + 1);
00072 //      printf("best list item %d:  %s\n", rnk, best_list[rnk]->get_code());
00073     }
00074
00075     delete [] best_list;
00076 }
```

6.66 FLAGR/src/ram/MC.cpp File Reference

6.67 MC.cpp

Go to the documentation of this file.

```
00001
00005
00006 void MergedList::MC(class InputList ** inlists, class SimpleScoreStats * s, class InputParams * prms)
      {
00007
00008     rank_t i = 0;
00009
00010     qsort(this->item_list, this->num_nodes, sizeof(class MergedItem *), &MergedList::cmp_code_asc);
00011
00012     score_t * state_matrix = this->compute_state_matrix(s, prms);
00013
00014     for (i = 0; i < this->num_nodes; i++) {
00015         this->item_list[i]->set_final_score( state_matrix[i] );
00016     }
00017
00018     delete [] state_matrix;
00019
00020     qsort(this->item_list, this->num_nodes, sizeof(class MergedItem *), &MergedList::cmp_score_desc);
00021
00022     //this->display_list();
00023 }
00024
00026 score_t * MergedList::compute_state_matrix(class SimpleScoreStats * s, class InputParams * prms) {
00027     class MergedItem *p, *q;
00028
00029     rank_t p_rank = 0, q_rank = 0, i = 0, j = 0, num_items = this->num_nodes;
00030     score_t val = 0.0, sum = 0.0, voter_weight = 1.0;
00031     score_t ergodic_number = prms->get_alpha();
00032
00033     uint32_t weights_norm = prms->get_weights_normalization();
00034     uint32_t ram = prms->get_aggregation_method();
00035     uint32_t num_lists = this->num_input_lists;
00036     uint32_t k = 0, wins = 0, lists_with_both = 0;
00037     int32_t iteration = 0;
00038
00040     score_t ** transition_matrix = new score_t * [ num_items ];
00041     for (i = 0; i < num_items; i++) {
00042         transition_matrix[i] = new score_t[ num_items ];
00043         if (!transition_matrix[i]) {
00044             printf("Could not allocate memory to accommodate the transition matrix.\nInput data is too
      large");
00045             exit(-1);
00046         }
00047     }
00048
00050     for (i = 0; i < num_items; i++) {
00051         p = this->item_list[i];
00052
00053         sum = 0.0;
00054         lists_with_both = 0;
00055 //      printf("[ ");
00056
00057         for (j = 0; j < num_items; j++) {
00058             q = this->item_list[j];
00059
00060             wins = 0;
00061
00062             for (k = 0; k < num_lists; k++) {
00063                 voter_weight = p->get_ranking(k)->get_input_list()->get_voter()->get_weight();
00064
00066                 if (weights_norm == 2) {
```

```
00067                     voter_weight = (voter_weight - s->get_min_val()) / (s->get_max_val() -
      s->get_min_val());
00068
00070                 } else if (weights_norm == 3) {
00071                     voter_weight = voter_weight * s->get_std_val() * s->get_std_val() /
      s->get_max_val();
00072
00074                 } else if (weights_norm == 4) {
00075                     voter_weight = voter_weight / s->get_max_val();
00076                 }
00077
00078                 p_rank = p->get_ranking(k)->get_rank();
00079                 q_rank = q->get_ranking(k)->get_rank();
00080
00081 //              printf("%s vs %s in List %d :  (%d - %d)\n", p->get_code(), q->get_code(), k, p_rank,
      q_rank);
00082                 if (p_rank != NOT_RANKED_ITEM_RANK && q_rank != NOT_RANKED_ITEM_RANK) {
00083                     lists_with_both++;
00084
00085                     if (p_rank < q_rank) {
00086                         wins++;
00087                     }
00088 //                  printf(" == Wins:  %d\n", wins);
00089                 }
00090             }
00091
00093             if (ram == 801) {
00094                 if (wins == num_lists) {
00095                     val = 0.0;
00096                 } else {
00097                     val = 1.0 / (score_t)num_items;
00098                 }
00099             }
00100
00102             if (ram == 802) {
00103                 if (wins > ((score_t)num_lists / 2.0)) {
00104                     val = 0.0;
00105                 } else {
00106                     val = 1.0 / (score_t)num_items;
00107                 }
00108             }
00109
00111             if (ram == 803) {
00112                 if (lists_with_both == 0) {
00113                     val = 0.0;
00114                 } else {
00115                     val = (score_t)wins / (score_t)(lists_with_both * num_items);
00116                 }
00117             }
00118
00120             if (ram == 804) {
00121                 if (lists_with_both == 0) {
00122                     val = 0.5 / (score_t)num_items;
00123                 } else {
00124                     if (wins > ((score_t)num_lists / 2.0)) {
00125                         val = 0.0;
00126                     } else {
00127                         val = 1.0 / (score_t)num_items;
00128                     }
00129                 }
00130             }
00131
00133             if (ram == 805) {
00134                 if (lists_with_both == 0) {
00135                     val = 0.5 / (score_t)num_items;
00136                 } else {
00137                     val = (lists_with_both - wins) / (score_t)(lists_with_both * num_items);
00138                 }
00139             }
00140
00142             if (i != j) {
00143                 transition_matrix[i][j] = val;
00144                 sum += val;
00145             }
00146 //          printf("%5.3f  ", transition_matrix[i][j]);
00147         }
00148
00150         transition_matrix[i][i] = 1.0 - sum;
00151
00152 //      printf("%5.3f  ", transition_matrix[i][i]);
00153 //      printf("]\n");
00154     }
00155
00156 //      getchar();
00159     for (i = 0; i < num_items; i++) {
00160 //      printf("[ ");
00161
```

```
00162         for (j = 0; j < num_items; j++) {
00163             transition_matrix[i][j] = transition_matrix[i][j] * (1 - ergodic_number) +
00164                     ergodic_number / (score_t)num_items;
00165
00166 //          printf("%5.4f  ", transition_matrix[i][j]);
00167         }
00168 //      printf("]\n");
00169     }
00170
00172     score_t * state_matrix = new score_t [ num_items ];
00173     score_t * next_state_matrix = new score_t [ num_items ];
00174     bool continue_loop = true;
00175
00176     for (i = 0; i < num_items; i++) {
00177         state_matrix[i] = 1.0 / (score_t)num_items;
00178         next_state_matrix[i] = 0.0;
00179     }
00180
00182     iteration = 1;
00183     while(iteration < prms->get_max_iterations()) {
00184
00185         this->matrixvec_multiply(next_state_matrix, state_matrix, transition_matrix);
00186
00187         continue_loop = false;
00188         for (i = 0; i < num_items; i++) {
00189             if ( abs(next_state_matrix[i] - state_matrix[i]) > prms->get_convergence_precision() ) {
00190                 continue_loop = true;
00191                 break;
00192             }
00193         }
00194
00195         if (!continue_loop) {
00196             break;
00197         }
00198
00199         for (i = 0; i < num_items; i++) {
00200             state_matrix[i] = next_state_matrix[i];
00201             next_state_matrix[i] = 0;
00202         }
00203
00204         iteration++;
00205     }
00206
00207 //  for (i = 0; i < num_items; i++) { printf("%7.6f\n", next_state_matrix[i]); }
00208
00210     for (i = 0; i < num_items; i++) {
00211         delete [] transition_matrix[i];
00212     }
00213     delete [] transition_matrix;
00214     delete [] next_state_matrix;
00215
00216     return state_matrix;
00217 }
00218
00219
00221 void MergedList::matrixvec_multiply(score_t * res_mat, score_t * state_mat, score_t ** tran_mat) {
00222
00223     rank_t i = 0, j = 0, num_items = this->num_nodes;
00224
00225     for (i = 0; i < num_items; i++) {
00226         for (j = 0; j < num_items; j++) {
00227             res_mat[i] += state_mat[j] * tran_mat[j][i];
00228         }
00229 //      printf("S[%d]=%5.6f\n", i, (res_mat)[i]);
00230     }
00231 }
```

6.68 FLAGR/src/ram/OutrankingApproach.cpp File Reference

6.69 OutrankingApproach.cpp

Go to the documentation of this file.

```
00001
00006
00007 void MergedList::Outranking(class InputList ** inlists, class SimpleScoreStats * s, class InputParams
      * prms) {
00008     score_t PREF_THRESHOLD = prms->get_pref_thr();
00009     score_t VETO_THRESHOLD = prms->get_veto_thr();
```

```
00010     score_t CONC_THRESHOLD = prms->get_conc_thr();
00011     score_t DISC_THRESHOLD = prms->get_disc_thr();
00012
00013     uint32_t preference_threshold = (uint32_t)(PREF_THRESHOLD * this->num_nodes);
00014     uint32_t veto_threshold = (uint32_t)(VETO_THRESHOLD * this->num_nodes);
00015     uint32_t concordance_threshold = (uint32_t)(CONC_THRESHOLD * this->num_input_lists);
00016     uint32_t discordance_threshold = (uint32_t)(DISC_THRESHOLD * this->num_input_lists);
00017
00018     class MergedItem *p, *q;
00019     score_t temp_score_1 = 0.0, temp_score_2 = 0.0, p_rank = 0.0, q_rank = 0.0;
00020     bool verbose = false;
00021     uint32_t scenario = 2;
00022
00023     score_t voter_weight = 0.0;
00024     uint32_t weights_norm = prms->get_weights_normalization();
00025
00026     for (rank_t i = 0; i < this->num_nodes; i++) {
00027         p = this->item_list[i];
00028
00029         for (rank_t j = 0; j < this->num_nodes; j++) {
00030             q = this->item_list[j];
00031
00032             temp_score_1 = 0.0;
00033             temp_score_2 = 0.0;
00034
00035             if (p != q) {
00036                 for (uint32_t k = 0; k < this->num_input_lists; k++) {
00037                     voter_weight = p->get_ranking(k)->get_input_list()->get_voter()->get_weight();
00038
00040                     if (weights_norm == 2) {
00041                         voter_weight = (voter_weight - s->get_min_val()) / (s->get_max_val() -
    s->get_min_val());
00042
00044                     } else if (weights_norm == 3) {
00045                         voter_weight = voter_weight * s->get_std_val() * s->get_std_val() /
    s->get_max_val();
00046
00048                     } else if (weights_norm == 4) {
00049                         voter_weight = voter_weight / s->get_max_val();
00050                     }
00051
00052                     if (scenario == 1) {
00053                         p_rank = p->get_ranking(k)->get_rank() * voter_weight;
00054                         q_rank = q->get_ranking(k)->get_rank() * voter_weight;
00055
00056                     } else if (scenario == 2) {
00057                         p_rank = p->get_ranking(k)->get_rank();
00058                         q_rank = q->get_ranking(k)->get_rank();
00059                     }
00060 //                  printf("%d:  %5.3f - %5.3f - (%5.3f, %5.3f)\n", k,
00061 //                      p->get_ranking(k)->get_input_list()->get_voter()->get_weight(),
00062 //                      q->get_ranking(k)->get_input_list()->get_voter()->get_weight(), p_rank,
    q_rank);
00063
00064                     if (p_rank <= q_rank - preference_threshold) {
00065                         if (scenario == 1) { temp_score_1++; }
00066                         else if (scenario == 2) { temp_score_1 += voter_weight; }
00067                     }
00068
00069                     if (p_rank >= q_rank + veto_threshold) {
00070                         if (scenario == 1) { temp_score_2 ++; }
00071                         else if (scenario == 2) { temp_score_2 += voter_weight; }
00072                     }
00073
00074                     if (verbose) {
00075                         printf("LIST %d:  Item %d (%s) rank %5.3f (VS) item %d (%s) rank %5.3f\n",
00076                             k, i, p->get_code(), p_rank, j, q->get_code(), q_rank);
00077                     }
00078                 }
00079             }
00080
00081             if (verbose) {
00082                 printf("Concordance = %5.2f.  Discordance = %5.2f\n", temp_score_1, temp_score_2);
00083                 getchar();
00084             }
00085
00087             if (temp_score_1 >= concordance_threshold && temp_score_2 <= discordance_threshold) {
00088                 if (scenario == 1) {
00089                     p->set_final_score(p->get_final_score() + 1.0);
00090                 } else if (scenario == 2) {
00091                     p->set_final_score(p->get_final_score() + temp_score_1);
00092                 }
00093             }
00094         }
00095     }
00096
00097     qsort(this->item_list, this->num_nodes, sizeof(class MergedItem *), &MergedList::cmp_score_desc);
```

```
00098 }
```

6.70 FLAGR/src/ram/PrefRel.cpp File Reference

```
#include "tools/MergedItemPair.cpp"
```

6.71 PrefRel.cpp

Go to the documentation of this file.

```
00001
00005
00006 #include "tools/MergedItemPair.cpp"
00007
00008 void MergedList::PrefRel(class InputList ** inlists, class SimpleScoreStats * s, class InputParams *
      prms) {
00009     uint32_t i = 0, j = 0, p = 0;
00010     uint32_t num_pairs = this->num_nodes * (this->num_nodes - 1) / 2;
00011     score_t DisagreementScore = 0.0, ListScore = 0.0, ItemScore = 0.0;
00012
00013     class InputList * inlist = NULL;
00014     class MergedItemPair * ItemPair = new MergedItemPair();
00015     char * key = NULL, * prev_key = NULL;
00016
00018     for (i = 0; i < this->num_nodes - 1; i++) {
00019         for (j = i + 1; j < this->num_nodes; j++) {
00020             ItemPair->set_item1( this->item_list[i] );
00021             ItemPair->set_item2( this->item_list[j] );
00022             ItemPair->compute_a_majority_opinion(prms->get_alpha(), prms->get_beta(),
      this->num_input_lists);
00023         }
00024     }
00025
00026     delete ItemPair;
00027
00029 //  printf("\n\nNum Nodes:  %d, Num Pairs:  %d\n", this->num_nodes, num_pairs);
00030     for (i = 0; i < this->num_input_lists; i++) {
00031         inlist = this->item_list[0]->get_ranking(i)->get_input_list();
00032         DisagreementScore = inlist->get_voter()->get_weight();
00033
00034         ListScore = 1.0 - DisagreementScore / num_pairs;
00035         inlist->set_voter_weight ( ListScore );
00036
00037 //      printf("List %d disagreement score = %2.1f, weight = %12.10f\n", i, DisagreementScore,
      ListScore);
00038     }
00039
00041     class MergedItemPair ** edges = new MergedItemPair * [2 * num_pairs];
00042     p = 0;
00043     for (i = 0; i < this->num_nodes - 1; i++) {
00044         for (j = i + 1; j < this->num_nodes; j++) {
00045             edges[p] = new MergedItemPair( this->item_list[i], this->item_list[j] );
00046             edges[p]->compute_weight();
00047             p++;
00048
00049             edges[p] = new MergedItemPair( this->item_list[j], this->item_list[i] );
00050             edges[p]->compute_weight();
00051             p++;
00052         }
00053     }
00054
00056     qsort(edges, 2 * num_pairs, sizeof(class MergedItemPait *), &MergedList::cmp_edges);
00057
00059     ItemScore = 0.0;
00060     for (i = 0; i < 2 * num_pairs; i++) {
00061         key = edges[i]->get_item2()->get_code();
00062         if (i == 0) {
00063             prev_key = key;
00064         }
00065
00066 //      edges[i]->display(0);
00067
00068         if (key != prev_key) {
00069 //          printf("\tItemScore for %s = %12.10f\n", prev_key, ItemScore);
00070             this->update_weight(prev_key, ItemScore);
```

```
00071             ItemScore = edges[i]->get_score();
00072             prev_key = key;
00073         } else {
00074             ItemScore += edges[i]->get_score();
00075         }
00076     }
00077
00078 //  printf("\tItemScore for %s = %12.10f\n", prev_key, ItemScore); getchar();
00079
00080     this->update_weight(prev_key, ItemScore);
00081
00082     qsort(this->item_list, this->num_nodes, sizeof(class MergedItem *), &MergedList::cmp_score_desc);
00083
00084     for (i = 0; i < 2 * num_pairs; i++) {
00085         delete edges[i];
00086     }
00087     delete [] edges;
00088 }
```

6.72 FLAGR/src/ram/RobustRA.cpp File Reference

6.73 RobustRA.cpp

Go to the documentation of this file.

```
00001
00005
00007 void MergedList::RobustRA(class InputList ** inlists,  class SimpleScoreStats * s, class InputParams *
      prms) {
00008     uint32_t i = 0;
00009     int error;
00010     rank_t rnk = 0;
00011     score_t min_p = 0.0;
00012     class MergedItem *p = NULL;
00013     class Ranking *r = NULL;
00014
00015     uint32_t num_elements = this->num_nodes;
00016 //  uint32_t num_elements = 6206; // for testing with RobustRankAggreg
00017
00019     for (rnk = 0; rnk < this->num_nodes; rnk++) {
00020         p = this->item_list[rnk];
00021
00022         for (i = 0; i < this->num_input_lists; i++) {
00023             r = p->get_ranking(i);
00024
00025             if (r->get_rank() == NOT_RANKED_ITEM_RANK) {
00026                 r->set_score(1.0);
00027             } else {
00028                 r->set_score(r->get_rank() / (double)num_elements);
00029             }
00030         }
00031     }
00032
00035     for (rnk = 0; rnk < this->num_nodes; rnk++) {
00036         p = this->item_list[rnk];
00037
00039         p->sort_rankings_by_score();
00040
00043         p->compute_beta_values();
00044     }
00045
00049     if (prms->get_exact()) {
00050         double * rm = new double[this->num_input_lists];
00051         double * buf = new double[this->num_input_lists + 2];
00052         double * buf2 = new double[this->num_input_lists + 2];
00053
00054         double * factorials170 = this->precompute_170_factorials();
00055
00056         for (rnk = 0; rnk < this->num_nodes; rnk++) {
00057             p = this->item_list[rnk];
00058
00059             min_p = 1.0;
00060             for (uint32_t j = 0; j < p->get_num_alloc_rankings(); j++) {
00061                 r = p->get_ranking(j);
00062                 if (r->get_score() < min_p) {
00063                     min_p = r->get_score();
00064                 }
00065             }
00066
```

```
00067             for (uint32_t j = 0; j < this->num_input_lists; j++) {
00068                 rm[this->num_input_lists - j - 1] = 1.0 - xinbta(this->num_input_lists - j, j + 1,
00069                         log(betaFunction(this->num_input_lists-j, j+1)), min_p, &error);
00070             }
00071 
00072             double score = 1.0 - this->stuart(rm, buf, buf2, factorials170, p->get_code());
00073 
00074             p->set_final_score( score );
00075         }
00076 
00077         delete [] factorials170;
00078         delete [] rm;
00079         delete [] buf;
00080         delete [] buf2;
00081 
00084     } else {
00085         for (rnk = 0; rnk < this->num_nodes; rnk++) {
00086             p = this->item_list[rnk];
00087 
00088             min_p = 1.0;
00089             for (uint32_t j = 0; j < p->get_num_alloc_rankings(); j++) {
00090                 r = p->get_ranking(j);
00091                 if (r->get_score() < min_p) {
00092                     min_p = r->get_score();
00093                 }
00094             }
00095 
00096             if (min_p * p->get_num_alloc_rankings() < 1.0) {
00097                 p->set_final_score( min_p * p->get_num_alloc_rankings() );
00098             } else {
00099                 p->set_final_score(1.0);
00100             }
00101         }
00102     }
00103 
00105     qsort(this->item_list, this->num_nodes, sizeof(class MergedItem *), &MergedList::cmp_score_asc);
00106 
00107 //  this->display_list();
00108 }
00109 
00110 
00112 score_t MergedList::stuart(double * r, double * v, double * p, double * factorials170, char * code) {
00113     score_t score = 0.0;
00114 
00115     qsort(r, this->num_input_lists, sizeof(double), &MergedList::cmp_double);
00116 
00118     for (uint32_t k = 0; k <= this->num_input_lists + 1; k++) {
00119         v[k] = 1.0;
00120         p[k] = 1.0;
00121     }
00122 
00123     for (uint32_t k = 1; k <= this->num_input_lists; k++) {
00124         v[k + 1] = this->sumStuart(v, r[this->num_input_lists - k], k, p, factorials170);
00125     }
00126 
00127     score = factorials170[this->num_input_lists] * v[this->num_input_lists + 1];
00128 
00129     return score;
00130 }
00131 
00132 
00133 score_t MergedList::sumStuart(double * v, double r, uint32_t k, double * p, double * factorials170) {
00134     for (uint32_t i = 1; i <= k; i++) {
00135         p[i] = v[k - i + 1] * pow(r, i) / factorials170[i];
00136 //      printf("%E, ", p[i]);
00137     }
00138 
00139 //  printf("\n==========\n");
00140     double sum = 0.0;
00141     for (uint32_t i = 1; i <= k; i++) {
00142         if (i % 2 == 0) {
00143             sum -= p[i];
00144         } else {
00145             sum += p[i];
00146         }
00147 //      printf("%E, ", sum);
00148     }
00149 
00150     return sum;
00151 }
```

6.74 FLAGR/src/ram/tools/BetaDistribution.cpp File Reference

Functions

- double betaFunction (uint32_t a, uint32_t b)
- double betain (double x, double p, double q, double beta, int *ifault)
- double pbeta (score_t x, uint32_t a, uint32_t b)

  Equivalent to the pbeta R function.
- double r8_max (double x, double y)
- double xinbta (double p, double q, double beta, double alpha, int *ifault)

6.74.1 Function Documentation

6.74.1.1 betaFunction()

```
double betaFunction (
            uint32_t a,
            uint32_t b )
```

The Beta Function is NOT implemented via the Gamma Function (tgamma) because tgamma tends to overflow even for moderate values. However, for integers we use the expression B(m,n) = (m-1)!(n-1)!/(m+n-1)! In the following implementation we do not compute the factorials individually (that would blow up too). Instead, we exploit the existence of the denominator and we compute B as a whole.

Definition at line 6 of file BetaDistribution.cpp.

6.74.1.2 betain()

```
double betain (
            double x,
            double p,
            double q,
            double beta,
            int * ifault )
```

ASA 063 Algorithm: https://people.math.sc.edu/Burkardt/c_src/asa063/betain.c BETAIN computes the incomplete Beta function ratio. Licensing: This code is distributed under the GNU LGPL license. Modified: 31 October 2010 Author: Original FORTRAN77 version by KL Majumder, GP Bhattacharjee. C version by John Burkardt. Reference: KL Majumder, GP Bhattacharjee, "Algorithm AS 63: The incomplete Beta Integral", Applied Statistics, Volume 22, Number 3, 1973, pages 409-411. Parameters: Input, double X, the argument, between 0 and 1. Input, double P, Q, the parameters, which must be positive. Input, double BETA, the logarithm of the complete beta function. Output, int *IFAULT, error flag. 0, no error. nonzero, an error occurred. Output, double BETAIN, the value of the incomplete Beta function ratio. Check the input arguments.

Special cases.

Change tail if necessary and determine S.

Use the Soper reduction formula.

Definition at line 41 of file BetaDistribution.cpp.

6.74.1.3 pbeta()

```
double pbeta (
            score_t x,
            uint32_t a,
            uint32_t b )
```

Equivalent to the pbeta R function.

Definition at line 136 of file BetaDistribution.cpp.

6.74.1.4 r8_max()

```
double r8_max (
            double x,
            double y )
```

R8_MAX returns the maximum of two R8's. Licensing: This code is distributed under the GNU LGPL license. Modified: 18 August 2004 Author: John Burkardt Parameters: Input, double X, Y, the quantities to compare. Output, double R8_MAX, the maximum of X and Y.

Definition at line 150 of file BetaDistribution.cpp.

6.74.1.5 xinbta()

```
double xinbta (
            double p,
            double q,
            double beta,
            double alpha,
            int * ifault )
```

ASA 109 Algorithm: https://people.math.sc.edu/Burkardt/c_src/asa109/asa109.c XINBTA computes inverse of the incomplete Beta function. Discussion: The accuracy exponent SAE was loosened from -37 to -30, because the code would not otherwise accept the results of an iteration with p = 0.3, q = 3.0, alpha = 0.2. Licensing: This code is distributed under the GNU LGPL license. Modified: 13 January 2017 Author: Original FORTRAN77 version by GW Cran, KJ Martin, GE Thomas. C version by John Burkardt. Reference: GW Cran, KJ Martin, GE Thomas, Remark AS R19 and Algorithm AS 109: A Remark on Algorithms AS 63: The Incomplete Beta Integral and AS 64: Inverse of the Incomplete Beta Integeral, Applied Statistics, Volume 26, Number 1, 1977, pages 111-114. Parameters: Input, double P, Q, the parameters of the incomplete Beta function. Input, double BETA, the logarithm of the value of the complete Beta function. Input, double ALPHA, the value of the incomplete Beta function. 0 <= ALPHA <= 1. Output, int *IFAULT, error flag. 0, no error occurred. nonzero, an error occurred. Output, double XINBTA, the argument of the incomplete Beta function which produces the value ALPHA. Local Parameters: Local, double SAE, accuracy is requested to about 10^SAE. Test for admissibility of parameters.

If answer is easy to determine, return immediately.

Change tail if necessary.

Calculate the initial approximation.

Solve for X by a modified Newton-Raphson method, using the function BETAIN.

Iteration loop.

Choose damping factor.

Check whether current estimate is acceptable. The change "VALUE = TX" was suggested by Ivan Ukhov.

Definition at line 182 of file BetaDistribution.cpp.

6.75 BetaDistribution.cpp

Go to the documentation of this file.

```
00001
00006 double betaFunction(uint32_t a, uint32_t b) {
00007     if (a == 1) { return 1.0 / (double)b; }
00008     if (b == 1) { return 1.0 / (double)a; }
00009
00010     uint32_t i = 0;
00011     double B = 1.0;
00012     if (a < b) {
00013         for (i = 2; i <= a; i++) {
00014             B *= (double)(i - 1.0) / (double)(i + b - 2.0);
00015         }
00016         B *= 1.0 / (a + b - 1.0);
00017     } else {
00018         for (i = 2; i <= b; i++) {
00019             B *= (double)(i - 1.0) / (double)(i + a - 2.0);
00020         }
00021         B *= 1.0 / (a + b - 1.0);
00022     }
00023     return B;
00024 //  return tgamma(a) * tgamma(b) / tgamma(a+b);
00025 }
00026
00041 double betain (double x, double p, double q, double beta, int *ifault) {
00042     double acu = 0.1E-14;
00043     double ai;
00044     double cx;
00045     int indx;
00046     int ns;
00047     double pp;
00048     double psq;
00049     double qq;
00050     double rx;
00051     double temp;
00052     double term;
00053     double value;
00054     double xx;
00055
00056     value = x;
00057     *ifault = 0;
00058
00060     if (p <= 0.0 || q <= 0.0) {
00061         *ifault = 1;
00062         return value;
00063     }
00064
00065     if (x < 0.0 || 1.0 < x) {
00066         *ifault = 2;
00067         return value;
00068     }
00069
00071     if (x == 0.0 || x == 1.0) {
00072         return value;
00073     }
00074
00076     psq = p + q;
00077     cx = 1.0 - x;
00078
00079     if (p < psq * x) {
00080         xx = cx;
00081         cx = x;
00082         pp = q;
00083         qq = p;
00084         indx = 1;
00085     } else {
00086         xx = x;
00087         pp = p;
00088         qq = q;
00089         indx = 0;
00090     }
00091
00092     term = 1.0;
00093     ai = 1.0;
00094     value = 1.0;
00095     ns = ( int ) ( qq + cx * psq );
00096
00098     rx = xx / cx;
00099     temp = qq - ai;
00100     if (ns == 0) {
00101         rx = xx;
00102     }
00103
00104     for ( ; ; ) {
```

```
00105         term = term * temp * rx / (pp + ai);
00106         value = value + term;;
00107         temp = fabs ( term );
00108 
00109         if (temp <= acu && temp <= acu * value) {
00110             value = value * exp ( pp * log ( xx ) + ( qq - 1.0 ) * log ( cx ) - beta ) / pp;
00111 
00112             if (indx) {
00113                 value = 1.0 - value;
00114             }
00115             break;
00116         }
00117 
00118         ai = ai + 1.0;
00119         ns = ns - 1;
00120 
00121         if ( 0 <= ns ) {
00122             temp = qq - ai;
00123             if (ns == 0) {
00124                 rx = xx;
00125             }
00126         } else {
00127             temp = psq;
00128             psq = psq + 1.0;
00129         }
00130     }
00131 
00132     return value;
00133 }
00134 
00136 double pbeta(score_t x, uint32_t a, uint32_t b) {
00137     double betalog = log(betaFunction(a, b));
00138     int32_t error;
00139     return betain(x, a, b, betalog, &error);
00140 }
00141 
00142 
00150 double r8_max (double x, double y) {
00151     double value;
00152 
00153     if (y < x) {
00154         value = x;
00155     } else {
00156         value = y;
00157     }
00158     return value;
00159 }
00160 
00161 
00182 double xinbta (double p, double q, double beta, double alpha, int *ifault) {
00183     double a;
00184     double acu;
00185     double adj;
00186     double fpu;
00187     double g;
00188     double h;
00189     int iex;
00190     int indx;
00191     double pp;
00192     double prev;
00193     double qq;
00194     double r;
00195     double s;
00196     double sae = -30.0;
00197     double sq;
00198     double t;
00199     double tx;
00200     double value;
00201     double w;
00202     double xin;
00203     double y;
00204     double yprev;
00205 
00206     fpu = pow (10.0, sae);
00207 
00208     *ifault = 0;
00209     value = alpha;
00210 
00212     if (p <= 0.0) {
00213         fprintf(stderr, "\n");
00214         fprintf(stderr, "XINBTA - Fatal error!\n");
00215         fprintf(stderr, "  P <= 0.\n");
00216         *ifault = 1;
00217         exit(1);
00218     }
00219 
00220     if (q <= 0.0) {
```

```
00221         fprintf(stderr, "\n");
00222         fprintf(stderr, "XINBTA - Fatal error!\n");
00223         fprintf(stderr, "  Q <= 0.\n");
00224         *ifault = 1;
00225         exit(1);
00226     }
00227
00228     if (alpha < 0.0 || 1.0 < alpha) {
00229         fprintf(stderr, "\n");
00230         fprintf(stderr, "XINBTA - Fatal error!\n");
00231         fprintf(stderr, "  ALPHA not between 0 and 1.\n");
00232         *ifault = 2;
00233         exit(1);
00234     }
00235
00237     if (alpha == 0.0) {
00238         value = 0.0;
00239         return value;
00240     }
00241
00242     if (alpha == 1.0) {
00243         value = 1.0;
00244         return value;
00245     }
00246
00248     if (0.5 < alpha) {
00249         a = 1.0 - alpha;
00250         pp = q;
00251         qq = p;
00252         indx = 1;
00253     } else {
00254         a = alpha;
00255         pp = p;
00256         qq = q;
00257         indx = 0;
00258     }
00259
00261     r = sqrt ( - log ( a * a ) );
00262
00263     y = r - ( 2.30753 + 0.27061 * r ) / ( 1.0 + ( 0.99229 + 0.04481 * r ) * r );
00264
00265     if (1.0 < pp && 1.0 < qq) {
00266         r = ( y * y - 3.0 ) / 6.0;
00267         s = 1.0 / ( pp + pp - 1.0 );
00268         t = 1.0 / ( qq + qq - 1.0 );
00269         h = 2.0 / ( s + t );
00270         w = y * sqrt ( h + r ) / h - ( t - s ) * ( r + 5.0 / 6.0 - 2.0 / ( 3.0 * h ) );
00271         value = pp / ( pp + qq * exp ( w + w ) );
00272     } else {
00273         r = qq + qq;
00274         t = 1.0 / ( 9.0 * qq );
00275         t = r * pow ( 1.0 - t + y * sqrt ( t ), 3 );
00276
00277         if (t <= 0.0) {
00278             value = 1.0 - exp ( ( log ( ( 1.0 - a ) * qq ) + beta ) / qq );
00279         } else {
00280             t = ( 4.0 * pp + r - 2.0 ) / t;
00281
00282             if (t <= 1.0) {
00283                 value = exp ( ( log ( a * pp ) + beta ) / pp );
00284             } else {
00285                 value = 1.0 - 2.0 / ( t + 1.0 );
00286             }
00287         }
00288     }
00289
00291     r = 1.0 - pp;
00292     t = 1.0 - qq;
00293     yprev = 0.0;
00294     sq = 1.0;
00295     prev = 1.0;
00296
00297     if (value < 0.0001) {
00298         value = 0.0001;
00299     }
00300
00301     if (0.9999 < value) {
00302         value = 0.9999;
00303     }
00304
00305     iex = r8_max ( - 5.0 / pp / pp - 1.0 / pow ( a, 0.2 ) - 13.0, sae );
00306
00307     acu = pow ( 10.0, iex );
00308
00310     for ( ; ; ) {
00311         y = betain(value, pp, qq, beta, ifault);
00312
```

```
00313         if (*ifault != 0) {
00314             fprintf(stderr, "\n");
00315             fprintf(stderr, "XINBTA - Fatal error!\n");
00316             fprintf(stderr, "  BETAIN returns IFAULT = %d.\n", *ifault);
00317             *ifault = 3;
00318             exit (1);
00319         }
00320 
00321         xin = value;
00322         y = ( y - a ) * exp ( beta + r * log ( xin ) + t * log ( 1.0 - xin ) );
00323 
00324         if (y * yprev <= 0.0) {
00325             prev = r8_max (sq, fpu);
00326         }
00327 
00328         g = 1.0;
00329 
00330         for ( ; ; ) {
00332             for ( ; ; ) {
00333                 adj = g * y;
00334                 sq = adj * adj;
00335 
00336                 if (sq < prev) {
00337                     tx = value - adj;
00338                     if (0.0 <= tx && tx <= 1.0) {
00339                         break;
00340                     }
00341                 }
00342                 g = g / 3.0;
00343             }
00344 
00346             if (prev <= acu || y * y <= acu) {
00347                 value = tx;
00348                 if (indx) {
00349                     value = 1.0 - value;
00350                 }
00351                 return value;
00352             }
00353 
00354             if (tx != 0.0 && tx != 1.0) {
00355                 break;
00356             }
00357 
00358             g = g / 3.0;
00359         }
00360 
00361         if (tx == value) {
00362             break;
00363         }
00364 
00365         value = tx;
00366         yprev = y;
00367     }
00368 
00369     if (indx) {
00370         value = 1.0 - value;
00371     }
00372 
00373     return value;
00374 }
```

6.76 FLAGR/src/ram/tools/MergedItemPair.cpp File Reference

```
#include "MergedItemPair.h"
```

6.77 MergedItemPair.cpp

Go to the documentation of this file.

```
00001 #include "MergedItemPair.h"
00002 
00005 MergedItemPair::MergedItemPair() :  item1(NULL), item2(NULL), score(0.0) { }
00006 
00008 MergedItemPair::MergedItemPair(class MergedItem * i1, class MergedItem * i2) :
00009     item1(i1), item2(i2), score(0.0) {
```

```
00010 }
00011
00013 MergedItemPair::~MergedItemPair() { }
00014
00015
00017 void MergedItemPair::compute_a_majority_opinion(score_t a, score_t b, uint32_t N) {
00018     uint32_t r = 0, r1 = 0, r2 = 0;
00019     uint32_t n0 = 0, n1 = 0;
00020     score_t DisagreementScore = 0.00;
00021     class InputList * inlist;
00022
00023 //  printf("\n\nComparing\n"); this->item1->display(); printf("\twith\n\t"); this->item2->display();
00024
00026     for (r = 0; r < this->item1->get_num_alloc_rankings(); r++) {
00027         if (this->item1->get_ranking(r)->get_rank() < this->item2->get_ranking(r)->get_rank()) {
00028             n0++;
00029         } else if (this->item1->get_ranking(r)->get_rank() > this->item2->get_ranking(r)->get_rank())
      {
00030             n1++;
00031         }
00032     }
00033
00035     for (r = 0; r < this->item1->get_num_alloc_rankings(); r++) {
00036
00037         r1 = this->item1->get_ranking(r)->get_rank();
00038         r2 = this->item2->get_ranking(r)->get_rank();
00039         DisagreementScore = 0.0;
00040
00041         if (r1 == NOT_RANKED_ITEM_RANK && r2 == NOT_RANKED_ITEM_RANK) {
00042             DisagreementScore = 0.5;
00043
00044         } else {
00045             if (n0 + n1 >= ceil(b * N)) {
00046
00048                 if (n0 < a * (n0 + n1)) {
00049                     if (r1 < r2) {
00050                         DisagreementScore = 1.0;
00051                     }
00052
00053                 } else if (n1 < a * (n0 + n1)) {
00054                     if (r1 > r2) {
00055                         DisagreementScore = 1.0;
00056                     }
00057                 }
00058             }
00059         }
00060
00061         if (DisagreementScore > 0) {
00062             inlist = this->item1->get_ranking(r)->get_input_list();
00063             inlist->set_voter_weight( inlist->get_voter()->get_weight() + DisagreementScore );
00064         }
00065     }
00066 //  getchar();
00067 }
00068
00070 void MergedItemPair::compute_a_majority_opinion_debug(score_t a, score_t b, uint32_t N) {
00071     uint32_t r = 0, r1 = 0, r2 = 0;
00072     uint32_t n0 = 0, n1 = 0;
00073     score_t DisagreementScore = 0.00;
00074     class InputList * inlist;
00075
00076     printf("\n\nComparing\n"); this->item1->display(); printf("\twith\n\t"); this->item2->display();
00077
00079     for (r = 0; r < this->item1->get_num_alloc_rankings(); r++) {
00080         if (this->item1->get_ranking(r)->get_rank() < this->item2->get_ranking(r)->get_rank()) {
00081             n0++;
00082         } else if (this->item1->get_ranking(r)->get_rank() > this->item2->get_ranking(r)->get_rank())
      {
00083             n1++;
00084         }
00085     }
00086
00088     for (r = 0; r < this->item1->get_num_alloc_rankings(); r++) {
00089         printf("\tChecking list %d:\n", r);
00090
00091         r1 = this->item1->get_ranking(r)->get_rank();
00092         r2 = this->item2->get_ranking(r)->get_rank();
00093         DisagreementScore = 0.0;
00094
00095         if (r1 == NOT_RANKED_ITEM_RANK && r2 == NOT_RANKED_ITEM_RANK) {
00096             printf("\t\tBoth items are not ranked.\n");
00097             DisagreementScore = 0.5;
00098         } else {
00099             if (n0 + n1 >= ceil(b * N)) {
00100                 printf("\t\t(1):  %d + %d >= %2.1f is satisfied.\n", n0, n1, ceil(b * N));
00101
00103                 if (n0 < a * (n0 + n1)) {
```

```
00104                     printf("\t\t(2):  %d < %2.1f * (%d + %d) is satisfied for n0.\n", n0, a, n0, n1);
00105
00106                     if (r1 > r2) {
00107                         printf("\t\tList %d agrees with a-majority\n", r);
00108                     } else if (r1 < r2) {
00109                         printf("\t\tList %d disagrees with a-majority\n", r);
00110                         DisagreementScore = 1.0;
00111                     }
00112
00113                 } else if (n1 < a * (n0 + n1)) {
00114
00115                     printf("\t\t(2):  %d < %2.1f * (%d + %d) is satisfied for n1.\n", n1, a, n0, n1);
00116                     if (r1 < r2) {
00117                         printf("\t\tList %d agrees with a-majority\n", r);
00118                     } else if (r1 > r2) {
00119                         printf("\t\tList %d disagrees with a-majority\n", r);
00120                         DisagreementScore = 1.0;
00121                     }
00122
00123                 } else {
00124                     printf("\t\t(2):  None of %d < %2.1f * (%d + %d) and %d < %2.1f * (%d + %d) is
      satisfied.\n",
00125                             n0, a, n0, n1, n1, a, n0, n1);
00126                 }
00127             } else {
00128                 printf("\t\t(1):  %d + %d >= %2.1f NOT satisfied.\n", n0, n1, ceil(b * N));
00129             }
00130         }
00131
00132         printf("\t\tList %d disagreement score:  %2.1f\n", r, DisagreementScore);
00133         if (DisagreementScore > 0) {
00134             inlist = this->item1->get_ranking(r)->get_input_list();
00135             inlist->set_voter_weight( inlist->get_voter()->get_weight() + DisagreementScore );
00136         }
00137     }
00138 //  getchar();
00139 }
00140
00143 void MergedItemPair::compute_weight() {
00144     uint32_t r = 0, r1 = 0, r2 = 0;
00145
00146     for (r = 0; r < this->item1->get_num_alloc_rankings(); r++) {
00147         r1 = this->item1->get_ranking(r)->get_rank();
00148         r2 = this->item2->get_ranking(r)->get_rank();
00149
00152         if (r2 < r1) {
00153             this->score += this->item2->get_ranking(r)->get_input_list()->get_voter()->get_weight();
00154         }
00155     }
00156
00157 //  this->display(0);
00158 }
00159
00161 void MergedItemPair::display(uint32_t t) {
00162     if (t == 0) {
00163         printf("Edge (%s, %s) Score = %12.10f\n", this->item1->get_code(), this->item2->get_code(),
      this->score);
00164     } else if (t == 1) {
00165         printf("Edge Score:  %12.10f\n", this->score);
00166         printf("Left Node:\n");
00167         this->item1->display();
00168         printf("\nRight Node:\n");
00169         this->item2->display();
00170     }
00171 }
00172
00174 class MergedItem * MergedItemPair::get_item1() { return this->item1; }
00175 class MergedItem * MergedItemPair::get_item2() { return this->item2; }
00176 score_t MergedItemPair::get_score() { return this->score; }
00177
00179 void MergedItemPair::set_item1(class MergedItem * v) { this->item1 = v; }
00180 void MergedItemPair::set_item2(class MergedItem * v) { this->item2 = v; }
```

6.78 FLAGR/src/ram/tools/MergedItemPair.h File Reference

Classes

- class MergedItemPair

6.79 MergedItemPair.h

Go to the documentation of this file.

```
00001 #ifndef MERGEDITEMPAIR_H
00002 #define MERGEDITEMPAIR_H
00003
00004
00005 class MergedItemPair {
00006     private:
00007         class MergedItem * item1;
00008         class MergedItem * item2;
00009         score_t score;
00010
00011     public:
00012         MergedItemPair();
00013         MergedItemPair(class MergedItem *, class MergedItem *);
00014         ~MergedItemPair();
00015
00016         void compute_a_majority_opinion(score_t a, score_t b, uint32_t N);
00017         void compute_a_majority_opinion_debug(score_t a, score_t b, uint32_t N);
00018         void compute_weight();
00019         void display(uint32_t);
00020
00022         class MergedItem * get_item1();
00023         class MergedItem * get_item2();
00024         score_t get_score();
00025
00027         void set_item1(class MergedItem *);
00028         void set_item2(class MergedItem *);
00029         void set_score(score_t);
00030 };
00031
00032 #endif // MERGEDITEMPAIR_H
```

6.80 FLAGR/src/Ranking.cpp File Reference

```
#include "Ranking.h"
```

6.81 Ranking.cpp

Go to the documentation of this file.

```
00001 #include "Ranking.h"
00002
00004 Ranking::Ranking(class InputList * l, rank_t r, score_t s) :
00005     input_list(l),
00006     rank(r),
00007     score(s) { }
00008
00010 Ranking::~Ranking() {
00011 }
00012
00014 void Ranking::display() {
00015     printf("\tList ID: %d - Rank:  %d, Score:  %6.5f\n",
00016         this->input_list->get_id(), this->rank, this->score);
00017 }
00018
00020 void Ranking::set_input_list(class InputList * v) { this->input_list = v; }
00021 void Ranking::set_rank(rank_t v) { this->rank = v; }
00022 void Ranking::set_score(score_t v) { this->score = v; }
00023
00025 class InputList * Ranking::get_input_list() { return this->input_list; }
00026 rank_t Ranking::get_rank() { return this->rank; }
00027 score_t Ranking::get_score() { return this->score; }
```

6.82 FLAGR/src/Ranking.h File Reference

Classes

- class Ranking

6.83 Ranking.h

Go to the documentation of this file.

```
00001 #ifndef RANKING_H
00002 #define RANKING_H
00003
00006
00007 class Ranking {
00008     private:
00009         class InputList * input_list;
00010         rank_t rank;
00011         score_t score;
00012
00013     public:
00014         Ranking(class InputList *, rank_t, score_t);
00015         ~Ranking();
00016
00017         void display();
00018
00020         void set_input_list(class InputList *);
00021         void set_rank(rank_t);
00022         void set_score(score_t);
00023
00025         class InputList * get_input_list();
00026         rank_t get_rank();
00027         score_t get_score();
00028 };
00029
00030 #endif // RANKING_H
```

6.84 FLAGR/src/Rel.cpp File Reference

```
#include "Rel.h"
```

Functions

- void reverse (char s[ ])
- void itoa_l (int32_t n, char s[ ])

  Convert an Integer to an Alphanumeric buffer.
- void itoa_lp (int32_t n, char s[ ])

  Convert an Integer to an Alphanumeric buffer of fixed size of 6 characters.
- float str_to_float (char ∗arr)

6.84.1 Function Documentation

6.84.1.1 itoa_l()

```
void itoa_l (
             int32_t n,
             char s[] )
```

Convert an Integer to an Alphanumeric buffer.

record sign

make n positive

generate digits in reverse order

get next digit

delete it

Definition at line 53 of file Rel.cpp.

6.84.1.2 itoa_lp()

```
void itoa_lp (
             int32_t n,
             char s[] )
```

Convert an Integer to an Alphanumeric buffer of fixed size of 6 characters.

Definition at line 71 of file Rel.cpp.

6.84.1.3 reverse()

```
void reverse (
             char s[] )
```

itoa version for Linux Systems to convert an integer number to a char variable.

Definition at line 42 of file Rel.cpp.

6.84.1.4 str_to_float()

```
float str_to_float (
             char * arr )
```

Definition at line 87 of file Rel.cpp.

6.85 Rel.cpp

Go to the documentation of this file.

```
00001 #include "Rel.h"
00002
00004 Rel::Rel() :  code(NULL), judgment(0), next(NULL) { }
00005
00007 Rel::Rel(char * c, uint32_t j) :  code(NULL), judgment(j), next(NULL) {
00008     this->copy_code(c);
00009 }
00010
00012 void Rel::copy_code(char * c) {
00013     if (c) {
00014         this->code = new char [strlen(c) + 1];
00015         strcpy(this->code, c);
00016     }
00017 }
00018
00020 Rel::~Rel() {
00021     if (this->code) {
00022         delete [] this->code;
00023     }
00024 }
00025
00026 void Rel::display() {
00027     printf("\t%s Relevance:  %d\n", this->code, this->judgment);
00028 }
00029
00031 inline void Rel::set_judgment(uint32_t v) { this->judgment = v; }
00032 inline void Rel::set_code(char * v) { this->copy_code(v); }
00033 inline void Rel::set_next(class Rel * v) { this->next = v; }
00034
```

```
00036 inline char * Rel::get_code() { return this->code; }
00037 inline uint32_t Rel::get_judgment() { return this->judgment; }
00038 inline class Rel * Rel::get_next() { return this->next; }
00039
00042 void reverse(char s[]) {
00043     int32_t c = 0, i = 0, j = 0;
00044
00045     for (i = 0, j = strlen(s)-1; i < j; i++, j--) {
00046         c = s[i];
00047         s[i] = s[j];
00048         s[j] = c;
00049     }
00050 }
00051
00053 void itoa_l(int32_t n, char s[]) {
00054     int32_t i = 0, sign = 0;
00055
00056     if ((sign = n) < 0) {
00057         n = -n;
00058     }
00059     i = 0;
00060     do {
00061         s[i++] = n % 10 + '0';
00062     } while ((n /= 10) > 0);
00063     if (sign < 0) {
00064         s[i++] = '-';
00065     }
00066     s[i] = '\0';
00067     reverse(s);
00068 }
00069
00071 void itoa_lp (int32_t n, char s[]) {
00072     char t[10];
00073     itoa_l(n, t);
00074
00075     int32_t i = 0, j = 0, len = strlen(t);
00076
00077     for (i = 0; i < 6 - len; i++) {
00078         s[i] = 48;
00079     }
00080
00081     for (j = i; j < 6; j++) {
00082         s[j] = t[j - i];
00083     }
00084     s[j] = 0;
00085 }
00086
00087 float str_to_float(char *arr){
00088     int i,j,flag;
00089     float val;
00090     char c;
00091     i=0;
00092     j=0;
00093     val=0;
00094     flag=0;
00095     while ((c = *(arr+i))!='\0'){
00096 //      if ((c<'0')||(c>'9')) return 0;
00097         if (c!='.'){
00098             val =(val*10)+(c-'0');
00099             if (flag == 1){
00100                 --j;
00101             }
00102         }
00103         if (c=='.'){ if (flag == 1) return 0; flag=1;}
00104         ++i;
00105     }
00106     val = val*pow(10,j);
00107     return val;
00108 }
```

6.86 FLAGR/src/Rel.h File Reference

Classes

- class Rel

6.87 Rel.h

Go to the documentation of this file.

```
00001 #ifndef REL_H
00002 #define REL_H
00003
00006
00007 class Rel {
00008     private:
00009         char * code;
00010         uint32_t judgment;
00011         class Rel * next;
00012
00013     private:
00014         void copy_code(char *);
00015
00016     public:
00017         Rel();
00018         Rel(char *, uint32_t);
00019         ~Rel();
00020
00021         void display();
00022
00024         void set_code(char *);
00025         void set_judgment(uint32_t);
00026         void set_next(class Rel *);
00027
00029         char * get_code();
00030         uint32_t get_judgment();
00031         class Rel * get_next();
00032 };
00033
00034 #endif // REL_H
```

6.88 FLAGR/src/Rels.cpp File Reference

```
#include "Rels.h"
```

6.89 Rels.cpp

Go to the documentation of this file.

```
00001 #include "Rels.h"
00002
00004 Rels::Rels() :  hash_table(NULL), mask(0), num_slots(0), num_nodes(0), num_chains(0) { }
00005
00007 Rels::Rels(uint32_t size) :
00008         hash_table(new Rel * [size]),
00009         mask(size - 1),
00010         num_slots(size),
00011         num_nodes(0),
00012         num_chains(0) {
00013
00014             for (uint32_t i = 0; i < size; i++) {
00015                 this->hash_table[i] = NULL;
00016             }
00017 }
00018
00020 Rels::~Rels() {
00021     class Rel * q;
00022     if (this->hash_table) {
00023         for (uint32_t i = 0; i < this->num_slots; i++) {
00024             while (this->hash_table[i] != NULL) {
00025                 q = this->hash_table[i]->get_next();
00026                 delete this->hash_table[i];
00027                 this->hash_table[i] = q;
00028             }
00029         }
00030         delete [] this->hash_table;
00031     }
00032 }
00033
```

```
00034
00036 void Rels::insert(char * n, uint32_t j) {
00038     uint32_t HashValue = this->djb2(n) & this->mask;
00039
00041     if (this->hash_table[HashValue] != NULL) {
00042         class Rel * q;
00043
00045         for (q = this->hash_table[HashValue]; q != NULL; q = q->get_next()) {
00046             if (strcmp(q->get_code(), n) == 0) {
00047                 return;
00048             }
00049         }
00050     } else {
00051         this->num_chains++;
00052     }
00053
00054     this->num_nodes++;
00055
00057     class Rel * record = new Rel(n, j);
00058
00060     record->set_next(this->hash_table[HashValue]);
00061     this->hash_table[HashValue] = record;
00062 }
00063
00064
00066 bool Rels::search(char * n, uint32_t * r) {
00068     uint32_t HashValue = this->djb2(n) & this->mask;
00069
00071     if (this->hash_table[HashValue] != NULL) {
00072         class Rel * q;
00073
00075         for (q = this->hash_table[HashValue]; q != NULL; q = q->get_next()) {
00076             if (strcmp(q->get_code(), n) == 0) {
00077                 *r = q->get_judgment();
00078                 return true;
00079             }
00080         }
00081     }
00082     *r = 0;
00083     return false;
00084 }
00085
00086
00088 inline uint32_t Rels::get_num_nodes() { return this->num_nodes; }
00089
00090 void Rels::display() {
00091     class Rel * q;
00092     for (uint32_t i = 0; i < this->num_slots; i++) {
00093         if (this->hash_table[i] != NULL) {
00094             for (q = this->hash_table[i]; q != NULL; q = q->get_next()) {
00095                 q->display();
00096 //              getchar();
00097             }
00098         }
00099     }
00100 }
00101
00103 uint32_t Rels::djb2(char * str) {
00104     unsigned long hash = 5381;
00105     int c;
00106
00107     while ((c = *str++))
00108         hash = ((hash << 5) + hash) + c; /* hash * 33 + c */
00109
00110     return hash;
00111 }
```

6.90 FLAGR/src/Rels.h File Reference

Classes

- class Rels

6.91 Rels.h

Go to the documentation of this file.

```
00001 #ifndef RELS_H
00002 #define RELS_H
00003
00006
00007 class Rels {
00008     private:
00009         class Rel ** hash_table;
00010
00011         uint32_t mask;
00012         uint32_t num_slots;
00013         uint32_t num_nodes;
00014         uint32_t num_chains;
00015
00016     private:
00017         uint32_t djb2(char *);
00018
00019     public:
00020         Rels();
00021         Rels(uint32_t);
00022         ~Rels();
00023
00024         void display();
00025         void insert(char *, uint32_t);
00026         bool search(char *, uint32_t *);
00027
00028         uint32_t get_num_nodes();
00029 };
00030
00031 #endif // RELS_H
```

6.92 FLAGR/src/SimpleScoreStats.cpp File Reference

```
#include "SimpleScoreStats.h"
```

6.93 SimpleScoreStats.cpp

Go to the documentation of this file.

```
00001 #include "SimpleScoreStats.h"
00002
00004 SimpleScoreStats::SimpleScoreStats() :  min_val(1000.0), max_val(0.0), mean_val(0.0), std_val(0.0) { }
00005
00007 SimpleScoreStats::~SimpleScoreStats() {
00008
00009 }
00010
00012 void SimpleScoreStats::display() {
00013     printf("Min Val:  %5.3f, Max Val:  %5.3f, Mean Val:  %5.3f, Std:  %5.3f\n",
00014         this->min_val, this->max_val, this->mean_val, this->std_val);
00015 }
00016
00018 score_t SimpleScoreStats::get_min_val() { return this->min_val; }
00019 score_t SimpleScoreStats::get_max_val() { return this->max_val; }
00020 score_t SimpleScoreStats::get_mean_val() { return this->mean_val; }
00021 score_t SimpleScoreStats::get_std_val() { return this->std_val; }
00022
00024 void SimpleScoreStats::set_min_val(score_t v) { this->min_val = v; }
00025 void SimpleScoreStats::set_max_val(score_t v) { this->max_val = v; }
00026 void SimpleScoreStats::set_mean_val(score_t v) { this->mean_val = v; }
00027 void SimpleScoreStats::set_std_val(score_t v) { this->std_val = v; }
```

6.94 FLAGR/src/SimpleScoreStats.h File Reference

Classes

- class SimpleScoreStats

6.95 SimpleScoreStats.h

Go to the documentation of this file.

```
00001 #ifndef SIMPLESCORESTATS_H
00002 #define SIMPLESCORESTATS_H
00003
00004
00005 class SimpleScoreStats {
00006     private:
00007         score_t min_val;
00008         score_t max_val;
00009         score_t mean_val;
00010         score_t std_val;
00011
00012     public:
00013         SimpleScoreStats();
00014         ~SimpleScoreStats();
00015
00016         void display();
00017
00019         score_t get_min_val();
00020         score_t get_max_val();
00021         score_t get_mean_val();
00022         score_t get_std_val();
00023
00025         void set_min_val(score_t);
00026         void set_max_val(score_t);
00027         void set_mean_val(score_t);
00028         void set_std_val(score_t);
00029 };
00030
00031 #endif // SIMPLESCORESTATS_H
```

6.96 FLAGR/src/Voter.cpp File Reference

```
#include "Voter.h"
```

6.97 Voter.cpp

Go to the documentation of this file.

```
00001 #include "Voter.h"
00002
00004 Voter::Voter() :  name(NULL), weight(0.0) { }
00005
00007 Voter::Voter(char * n, score_t w) :  name(NULL), weight(w) {
00008     this->set_name(n);
00009 }
00010
00012 Voter::~Voter() {
00013     if (this->name) {
00014         delete [] this->name;
00015     }
00016 }
00017
00019 void Voter::display() {
00020     printf("Voter Name:  %s - Weight:  %5.3f\n", this->name, this->weight);
00021 }
00022
00024 void Voter::set_weight(score_t v) { this->weight = v; }
00025 void Voter::set_name(char * v) {
00026     this->name = new char[strlen(v) + 1];
00027     strcpy(this->name, v);
00028 }
00029
00031 char * Voter::get_name() { return this->name; }
00032 score_t Voter::get_weight() { return this->weight; }
```

6.98 FLAGR/src/Voter.h File Reference

Classes

- class Voter

6.99 Voter.h

Go to the documentation of this file.

```
00001 #ifndef VOTER_H
00002 #define VOTER_H
00003
00004
00005 class Voter {
00006     private:
00007         char * name;
00008         score_t weight;
00009
00010     public:
00011         Voter();
00012         Voter(char *, score_t);
00013         ~Voter();
00014
00015         void display();
00016
00018         void set_name(char *);
00019         void set_weight(score_t);
00020
00022         char * get_name();
00023         score_t get_weight();
00024 };
00025
00026 #endif // VOTER_H
```

Index