PyFLAGR

Generated by Doxygen 1.9.5

Chapter 1

(Py)FLAGR

∗∗Fuse, Learn, AGgregate, Rerank

FLAGR is a high performing, modular library for rank aggregation. To ensure the highest possible performance, the core FLAGR library is written in C++ and implements a wide collection of unsupervised rank aggregation methods. Its modular design allows third-party programmers to implement their own algorithms and easily rebuild the entire library. FLAGR can be built as a standard application, or as a shared library (`so` or `dll`). In the second case, it can be linked from other C/C++ programs, or even from programs written in other languages (e.g. Python, PHP, etc.).

In this context, PyFLAGR is a Python library that links to FLAGR and allows a developer to exploit the efficient FLAGR implementations from a standard Python program.

1.1 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 .
```

1.2 Importing and using PyFLAGR

PyFLAGR groups its supported rank aggregation methods in four modules:

1. `Comb`: In this module the `CombSUM` and `CombMNZ` methods are implemented. Each method comes in four variants according to the rank/score normalization method. Future releases of FLAGR will also include CombAVG, CombMAX and CombMIN.
2. `Majoritarian`: Includes `CondorcetWinners`, `CopelandWinners` and `Outranking↩ Approach`.
3. `MarkovChains`: The fourth and most popular method (termed `MC4`) based on Markov Chains is implemented. Future releases of FLAGR will include the other three implementations.

4. `Weighted`: This module implements several self-weighting rank aggregation methods. These methods automatically identify the expert voters and include:
   (a) The Preference Relations Graph method of Desarkar et.al, 2016.
   (b) The Agglomerative method of Chatterjee et.al, 2018.
   (c) The Iterative, Distance-Based method of Akritidis et.al, 2022.

The following statements demonstrate the imports of all PyFLAGR rank aggregation methods in a typical jupyter notebook.

```
import pyflagr.Comb as SCORE_BASED
import pyflagr.Majoritarian as ORDER_BASED
import pyflagr.MarkovChains as MARKOV_CHAINS
import pyflagr.Weighted as WGT
```

All PyFLAGR rank aggregation methods include:

- a standard class constructor: several hyper-parameters of the corresponding algorithm and other execution arguments can be passed through the constructor. All the constructor inputs have default values, therefore, they are considered optional. This means that all constructors can be called *any* argument at all.
- an `aggregate` method that runs the algorithm on the selected input and (optionally) evaluates the generated aggregate list. In all algorithms, `aggregate` method accepts the following arguments:

Parameter	Type	Default Value	Values
`input_file`	String - Required, unless `input_df` is set.	Empty String	A CSV file that contains the input lists to be aggregated.
`input_df`	Pandas DataFrame - Required, unless `input_file` is set.	`None`	A Pandas DataFrame that contains the input lists to be aggregated. **Note:** If both `input_file` and `input_df` are set, only the former is used; the latter is ignored.
`rels_file`	String, Optional.	Empty String	A CSV file that contains the relevance judgements of the involved list elements. If such a file is passed, FLAGR will evaluate the generated aggregate list/s by computing several retrieval effectiveness evaluation measures. The results of the evaluation will be stored in the `eval_df` DataFrame. Otherwise, no evaluation will take place and `eval_df` will be empty. Read more on the evaluation of rank aggregation quality.

Parameter	Type	Default Value	Values
rels_df	Pandas DataFrame, Optional.	None	A Pandas DataFrame that contains the relevance judgements of the involved list elements. If such a dataframe is passed, FLAGR will evaluate the generated aggregate list/s by computing several retrieval effectiveness evaluation measures. The results of the evaluation will be stored in the eval_df DataFrame. Otherwise, no evaluation will take place and eval_df will be empty. Read more on the evaluation of rank aggregation quality. **Note:** If both rels_file and rels_df are set, only the former is used; the latter is ignored.
output_dir	String, Optional.	Temporary directory (OS-specific)	The directory where the output files (aggregate lists and evaluation) will be stored. If it is not set, the default location will be used.

1.3 Input data

The core library, FLAGR, accepts data (namely, the input lists to be aggregated) in a single, specially formatted CSV file. The columns in the CSV file are organized according to the following manner:

```
Query/Topic,Voter,Item,Score,Algorithm/Dataset
```

where:

- Query/Topic: the query string or the topic for which the list is submitted.
- Voter: the name of the voter, or the ranker who submits the list.
- Item: a unique name that identifies a particular element in the list. A voter cannot submit the same element for the same query/topic two or more times. This means that each element appears exactly once in each list. However, the same element may appear in lists submitted by other voters.
- Score: the score assigned to an Item by a specific Voter. In may cases (e.g. search engine rankings), the individual scores are unknown. In such cases the scores can be replaced by the (reverse) ranking of an Item in such a manner that the top rankings receive higher scores than the ones that have been assigned lower rankings.

PyFLAGR has two mechanisms for passing data to FLAGR, namely:

- either by forwarding the name and the location of the aforementioned input CSV file (this is the input_file argument of the aggregate method),

- or by accepting a Pandas Dataframe from the user (this is the `input_df` argument of the `aggregate` method). In this case, PyFLAGR internally dumps the `input_df` contents into a temporary CSV file and passes the name and the location of that temporary file to FLAGR.

Optionally, the user may specify a second CSV file (called as `rels_file`), or a Dataframe (called as `rels_df`) that contain judgments about the relevance of the included elements w.r.t a query. The columns in `rels_file` are organized as follows:

```
Query/Topic,0,Item,Relevance
```

where:

- `Query/Topic`: the query string or the topic for which the corresponding `Item` is evaluated.
- `0`: A hypothetical hyper-voter (also called voter `0`) who has flawless knowledge of the `Query/Topic` and determines whether an `Item` is relevant to it, or not. The value of this column must be always `0`.
- `Item`: a unique name that identifies a particular element of which the relevance to the `Query/Topic` is evaluated.
- `Relevance`: the relevance score assigned to an `Item` by `Voter 0`.

1.4 Output data format

PyFLAGR returns a Pandas Dataframe that contains the final aggregate list.

Optionally, FLAGR may also create a second output file to write the results of the evaluation of the effectiveness of an algorithm. This happens when a `rels_file` is provided to the algorithm. The `aggregate` method of all algorithms *always* returns two Pandas Dataframes according to the provided input.

1.5 References:

[1] 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.

[2] 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.

[3] 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.

[4] Chatterjee, S., Mukhopadhyay, A., Bhattacharyya, M., "A weighted rank aggregation approach towards crowd opinion analysis", Knowledge-Based Systems, vol. 149, pp. 47-60, 2018.

[5] 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.

[6] 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.

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

Chapter 4

Class Index

4.1 Class List

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

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

Namespace Documentation

6.1 pyflagr Namespace Reference

Namespaces

- namespace Comparator
- namespace Kemeny
- namespace Linear
- namespace Majoritarian
- namespace MarkovChains
- namespace RAM
- namespace RRA
- namespace test_local
- namespace Weighted

6.2 pyflagr.Comparator Namespace Reference

Classes

- class Comparator

6.3 pyflagr.Kemeny Namespace Reference

Classes

- class KemenyOptimal

6.4 pyflagr.Linear Namespace Reference

Classes

- class BordaCount
- class CombMNZ
- class CombSUM
- class SimpleBordaCount

6.5 pyflagr.Majoritarian Namespace Reference

Classes

- class CondorcetWinners
- class CopelandWinners
- class OutrankingApproach

6.6 pyflagr.MarkovChains Namespace Reference

Classes

- class MC
- class MC1
- class MC2
- class MC3
- class MC4
- class MCT

Variables

- float def_ergodic_number = 0.15
- int def_max_iterations = 100

6.6.1 Variable Documentation

6.6.1.1 def_ergodic_number

```
float pyflagr.MarkovChains.def_ergodic_number = 0.15
```

Definition at line 6 of file MarkovChains.py.

6.6.1.2 def_max_iterations

```
int pyflagr.MarkovChains.def_max_iterations = 100
```

Definition at line 7 of file MarkovChains.py.

6.7 pyflagr.RAM Namespace Reference

Classes

- class RAM

6.8 pyflagr.RRA Namespace Reference

Classes

- class RRA

6.9 pyflagr.test_local Namespace Reference

Variables

- string base_path = ''
- string lists = base_path + 'MOSO.csv'
- string qrels = base_path + 'MOSO_qrels.csv'
- input_dataframe = pd.read_csv(lists)
- rels_dataframe = pd.read_csv(qrels)
- int EV_PTS = 10
- cmp = Comparator.Comparator(EV_PTS)
- input_file
- rels_file

6.9.1 Variable Documentation

6.9.1.1 base_path

```
string pyflagr.test_local.base_path = ''
```

Definition at line 22 of file test_local.py.

6.9.1.2 cmp

```
pyflagr.test_local.cmp = Comparator.Comparator(EV_PTS)
```

Definition at line 64 of file test_local.py.

6.9.1.3 EV_PTS

```
int pyflagr.test_local.EV_PTS = 10
```

Definition at line 62 of file test_local.py.

6.9.1.4 input_dataframe

```
pyflagr.test_local.input_dataframe = pd.read_csv(lists)
```

Definition at line 33 of file test_local.py.

6.9.1.5 input_file

```
pyflagr.test_local.input_file
```

Definition at line 89 of file test_local.py.

6.9.1.6 lists

```
pyflagr.test_local.lists = base_path + 'MOSO.csv'
```

Definition at line 30 of file test_local.py.

6.9.1.7 qrels

```
string pyflagr.test_local.qrels = base_path + 'MOSO_qrels.csv'
```

Definition at line 31 of file test_local.py.

6.9.1.8 rels_dataframe

```
pyflagr.test_local.rels_dataframe = pd.read_csv(qrels)
```

Definition at line 34 of file test_local.py.

6.9.1.9 rels_file

```
pyflagr.test_local.rels_file
```

Definition at line 89 of file test_local.py.

6.10 pyflagr.Weighted Namespace Reference

Classes

- class Agglomerative
- class DIBRA
- class PreferenceRelationsGraph

6.11 setup Namespace Reference

Variables

- string DESCRIPTION = 'PyFLAGR is a Python package for aggregating ranked preference lists from multiple sources.'
- string LONG_DESCRIPTION
- name
- version
- description
- long_description
- long_description_content_type
- author
- author_email
- maintainer
- maintainer_email
- packages
- url
- install_requires
- license
- keywords
- py_modules
- package_data

6.11.1 Variable Documentation

6.11.1.1 author

```
setup.author
```

Definition at line 23 of file setup.py.

6.11.1.2 author_email

```
setup.author_email
```

Definition at line 24 of file setup.py.

6.11.1.3 DESCRIPTION

```
string setup.DESCRIPTION = 'PyFLAGR is a Python package for aggregating ranked preference
lists from multiple sources.'
```

Definition at line 4 of file setup.py.

6.11.1.4 description

```
setup.description
```

Definition at line 20 of file setup.py.

6.11.1.5 install_requires

```
setup.install_requires
```

Definition at line 29 of file setup.py.

6.11.1.6 keywords

```
setup.keywords
```

Definition at line 31 of file setup.py.

6.11.1.7 license

```
setup.license
```

Definition at line 30 of file setup.py.

6.11.1.8 LONG_DESCRIPTION

```
string setup.LONG_DESCRIPTION
```

Initial value:

```
00001 =  'The fusion of multiple ranked lists of elements into a single aggregate list is a well-studied '\
00002     'research field with numerous applications in Bioinformatics, recommendation systems,
      collaborative filtering, '\
00003     'election systems and metasearch engines.\n\n' \
00004     'FLAGR is a high performance, modular, open source library for rank aggregation problems.  It
      implements baseline '\
00005     'and recent state-of-the-art aggregation algorithms that accept ranked preference lists and
      generate a single '\
00006     'consensus list of elements.  A portion of these methods apply exploratory analysis techniques and
      belong to the '\
00007     'broad family of unsupervised learning techniques.\n\n' \
00008     'PyFLAGR is a Python library built on top of FLAGR library core.  It can be easily installed with
      pip and used in '\
00009     'standard Python programs and Jupyter notebooks.\n\n\n' \
00010     'FLAGR Website:  [https://flagr.site/](https://flagr.site/)\n\n' \
00011     'GitHub repository:
      [https://github.com/lakritidis/FLAGR](https://github.com/lakritidis/FLAGR)\n\n'
```

Definition at line 5 of file setup.py.

6.11.1.9 long_description

```
setup.long_description
```

Definition at line 21 of file setup.py.

6.11.1.10 long_description_content_type

```
setup.long_description_content_type
```

Definition at line 22 of file setup.py.

6.11.1.11 maintainer

```
setup.maintainer
```

Definition at line 25 of file setup.py.

6.11.1.12 maintainer_email

```
setup.maintainer_email
```

Definition at line 26 of file setup.py.

6.11.1.13 name

```
setup.name
```

Definition at line 18 of file setup.py.

6.11.1.14 package_data

```
setup.package_data
```

Definition at line 35 of file setup.py.

6.11.1.15 packages

```
setup.packages
```

Definition at line 27 of file setup.py.

6.11.1.16 py_modules

```
setup.py_modules
```

Definition at line 34 of file setup.py.

6.11.1.17 url

```
setup.url
```

Definition at line 28 of file setup.py.

6.11.1.18 version

```
setup.version
```

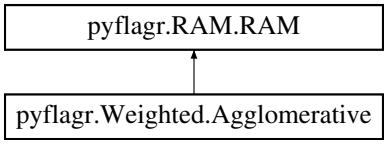
Definition at line 19 of file setup.py.

Chapter 7

Class Documentation

7.1 pyflagr.Weighted.Agglomerative Class Reference

Inheritance diagram for pyflagr.Weighted.Agglomerative:



Public Member Functions

- def __init__ (self, eval_pts=10, c1=C1, c2=C2)
- def aggregate (self, input_file="", input_df=None, rels_file="", rels_df=None, out_dir=None)

Public Attributes

- C1
- C2
- output_dir

Static Public Attributes

- float C1 = 0.1
- float C2 = 0.5

7.1.1 Detailed Description

Definition at line 62 of file Weighted.py.

7.1.2 Constructor & Destructor Documentation

7.1.2.1 __init__()

```
def pyflagr.Weighted.Agglomerative.__init__ (
              self,
              eval_pts = 10,
              c1 = C1,
              c2 = C2 )
```

Reimplemented from pyflagr.RAM.RAM.

Definition at line 66 of file Weighted.py.

7.1.3 Member Function Documentation

7.1.3.1 aggregate()

```
def pyflagr.Weighted.Agglomerative.aggregate (
              self,
              input_file = "",
              input_df = None,
              rels_file = "",
              rels_df = None,
              out_dir = None )
```

Definition at line 84 of file Weighted.py.

7.1.4 Member Data Documentation

7.1.4.1 C1 [1/2]

```
float pyflagr.Weighted.Agglomerative.C1 = 0.1  [static]
```

Definition at line 63 of file Weighted.py.

7.1.4.2 C1 [2/2]

```
pyflagr.Weighted.Agglomerative.C1
```

Definition at line 69 of file Weighted.py.

7.1.4.3 C2 [1/2]

```
float pyflagr.Weighted.Agglomerative.C2 = 0.5  [static]
```

Definition at line 64 of file Weighted.py.

7.1.4.4 C2 [2/2]

```
pyflagr.Weighted.Agglomerative.C2
```

Definition at line 70 of file Weighted.py.

7.1.4.5 output_dir

```
pyflagr.Weighted.Agglomerative.output_dir
```

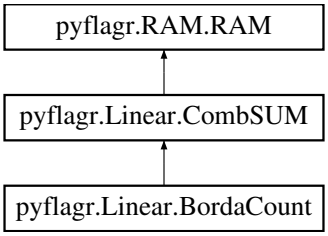
Definition at line 88 of file Weighted.py.

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

- PYFLAGR/pyflagr/Weighted.py

7.2 pyflagr.Linear.BordaCount Class Reference

Inheritance diagram for pyflagr.Linear.BordaCount:



Public Member Functions

- def __init__ (self, eval_pts=10)

Additional Inherited Members

7.2.1 Detailed Description

Definition at line 71 of file Linear.py.

7.2.2 Constructor & Destructor Documentation

7.2.2.1 __init__()

```
def pyflagr.Linear.BordaCount.__init__ (
              self,
              eval_pts = 10 )
```

Reimplemented from pyflagr.Linear.CombSUM.

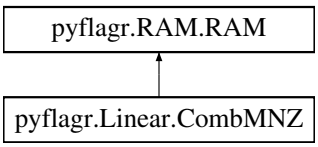
Definition at line 72 of file Linear.py.

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

- PYFLAGR/pyflagr/Linear.py

7.3 pyflagr.Linear.CombMNZ Class Reference

Inheritance diagram for pyflagr.Linear.CombMNZ:



Public Member Functions

- def __init__ (self, norm="borda", eval_pts=10)
- def aggregate (self, input_file="", input_df=None, rels_file="", rels_df=None, out_dir=None)

Public Attributes

- normalization
- output_dir

Static Public Attributes

- string normalization = "borda"

7.3.1 Detailed Description

Definition at line 83 of file Linear.py.

7.3.2 Constructor & Destructor Documentation

7.3.2.1 __init__()

```
def pyflagr.Linear.CombMNZ.__init__ (
              self,
              norm = "borda",
              eval_pts = 10 )
```

Reimplemented from pyflagr.RAM.RAM.

Definition at line 86 of file Linear.py.

7.3.3 Member Function Documentation

7.3.3.1 aggregate()

```
def pyflagr.Linear.CombMNZ.aggregate (
              self,
              input_file = "",
              input_df = None,
              rels_file = "",
              rels_df = None,
              out_dir = None )
```

Definition at line 100 of file Linear.py.

7.3.4 Member Data Documentation

7.3.4.1 normalization [1/2]

```
string pyflagr.Linear.CombMNZ.normalization = "borda"  [static]
```

Definition at line 84 of file Linear.py.

7.3.4.2 normalization [2/2]

```
pyflagr.Linear.CombMNZ.normalization
```

Definition at line 89 of file Linear.py.

7.3.4.3 output_dir

```
pyflagr.Linear.CombMNZ.output_dir
```

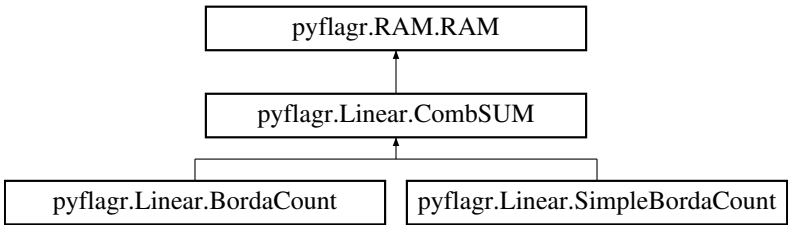
Definition at line 104 of file Linear.py.

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

- PYFLAGR/pyflagr/Linear.py

7.4 pyflagr.Linear.CombSUM Class Reference

Inheritance diagram for pyflagr.Linear.CombSUM:



Public Member Functions

- def __init__ (self, norm="borda", eval_pts=10)
- def aggregate (self, input_file="", input_df=None, rels_file="", rels_df=None, out_dir=None)

Public Attributes

- normalization
- output_dir

Static Public Attributes

- string normalization = "borda"

7.4.1 Detailed Description

Definition at line 8 of file Linear.py.

7.4.2 Constructor & Destructor Documentation

7.4.2.1 __init__()

```
def pyflagr.Linear.CombSUM.__init__ (
              self,
              norm = "borda",
              eval_pts = 10 )
```

Reimplemented from pyflagr.RAM.RAM.

Reimplemented in pyflagr.Linear.BordaCount, and pyflagr.Linear.SimpleBordaCount.

Definition at line 11 of file Linear.py.

7.4.3 Member Function Documentation

7.4.3.1 aggregate()

```
def pyflagr.Linear.CombSUM.aggregate (
              self,
              input_file = "",
              input_df = None,
              rels_file = "",
              rels_df = None,
              out_dir = None )
```

Definition at line 25 of file Linear.py.

7.4.4 Member Data Documentation

7.4.4.1 normalization [1/2]

```
string pyflagr.Linear.CombSUM.normalization = "borda"  [static]
```

Definition at line 9 of file Linear.py.

7.4.4.2 normalization [2/2]

```
pyflagr.Linear.CombSUM.normalization
```

Definition at line 14 of file Linear.py.

7.4.4.3 output_dir

```
pyflagr.Linear.CombSUM.output_dir
```

Definition at line 29 of file Linear.py.

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

- PYFLAGR/pyflagr/Linear.py

7.5 pyflagr.Comparator.Comparator Class Reference

Public Member Functions

- def __init__ (self, evaluation_points)
- def add_aggregator (self, name, obj)
- def aggregate (self, input_file="", input_df=None, rels_file="", rels_df=None)
- def plot_average_precision (self, dimensions, show_grid, query='all')
- def plot_metric (self, cutoff, metric, plot_type='bar', dimensions=(10.24, 7.68), show_grid=True, query='all')
- def get_df_slice (self, cutoff, metric, df)
- def get_results (self, cutoff, metric='all', query='all')
- def convert_to_latex (self, cutoff, metric='all', query='all', dec_pts=6)

Public Attributes

- ev_pts

Static Public Attributes

- aggregators = None
- results = None
- int ev_pts = 0

7.5.1 Detailed Description

Definition at line 6 of file Comparator.py.

7.5.2 Constructor & Destructor Documentation

7.5.2.1 __init__()

```
def pyflagr.Comparator.Comparator.__init__ (
              self,
              evaluation_points )
```

Definition at line 11 of file Comparator.py.

7.5.3 Member Function Documentation

7.5.3.1 add_aggregator()

```
def pyflagr.Comparator.Comparator.add_aggregator (
              self,
              name,
              obj )
```

Definition at line 17 of file Comparator.py.

7.5.3.2 aggregate()

```
def pyflagr.Comparator.Comparator.aggregate (
              self,
              input_file = "",
              input_df = None,
              rels_file = "",
              rels_df = None )
```

Definition at line 22 of file Comparator.py.

7.5.3.3 convert_to_latex()

```
def pyflagr.Comparator.Comparator.convert_to_latex (
              self,
              cutoff,
              metric = 'all',
              query = 'all',
              dec_pts = 6 )
```

Definition at line 116 of file Comparator.py.

7.5.3.4 get_df_slice()

```
def pyflagr.Comparator.Comparator.get_df_slice (
              self,
              cutoff,
              metric,
              df )
```

Definition at line 74 of file Comparator.py.

7.5.3.5 get_results()

```
def pyflagr.Comparator.Comparator.get_results (
              self,
              cutoff,
              metric = 'all',
              query = 'all' )
```

Definition at line 101 of file Comparator.py.

7.5.3.6 plot_average_precision()

```
def pyflagr.Comparator.Comparator.plot_average_precision (
              self,
              dimensions,
              show_grid,
              query = 'all' )
```

Definition at line 44 of file Comparator.py.

7.5.3.7 plot_metric()

```
def pyflagr.Comparator.Comparator.plot_metric (
              self,
              cutoff,
              metric,
              plot_type = 'bar',
              dimensions = (10.24, 7.68),
              show_grid = True,
              query = 'all' )
```

Definition at line 56 of file Comparator.py.

7.5.4 Member Data Documentation

7.5.4.1 aggregators

```
pyflagr.Comparator.Comparator.aggregators = None  [static]
```

Definition at line 7 of file Comparator.py.

7.5.4.2 ev_pts [1/2]

```
int pyflagr.Comparator.Comparator.ev_pts = 0  [static]
```

Definition at line 9 of file Comparator.py.

7.5.4.3 ev_pts [2/2]

```
pyflagr.Comparator.Comparator.ev_pts
```

Definition at line 14 of file Comparator.py.

7.5.4.4 results

```
pyflagr.Comparator.Comparator.results = None  [static]
```

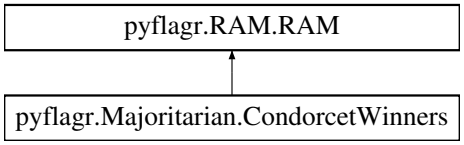
Definition at line 8 of file Comparator.py.

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

- PYFLAGR/pyflagr/Comparator.py

7.6 pyflagr.Majoritarian.CondorcetWinners Class Reference

Inheritance diagram for pyflagr.Majoritarian.CondorcetWinners:



Public Member Functions

- def __init__ (self, eval_pts=10)
- def aggregate (self, input_file="", input_df=None, rels_file="", rels_df=None, out_dir=None)

Public Attributes

- output_dir

Additional Inherited Members

7.6.1 Detailed Description

Definition at line 8 of file Majoritarian.py.

7.6.2 Constructor & Destructor Documentation

7.6.2.1 __init__()

```
def pyflagr.Majoritarian.CondorcetWinners.__init__ (
              self,
              eval_pts = 10 )
```

Reimplemented from pyflagr.RAM.RAM.

Definition at line 10 of file Majoritarian.py.

7.6.3 Member Function Documentation

7.6.3.1 aggregate()

```
def pyflagr.Majoritarian.CondorcetWinners.aggregate (
              self,
              input_file = "",
              input_df = None,
              rels_file = "",
              rels_df = None,
              out_dir = None )
```

Definition at line 23 of file Majoritarian.py.

7.6.4 Member Data Documentation

7.6.4.1 output_dir

```
pyflagr.Majoritarian.CondorcetWinners.output_dir
```

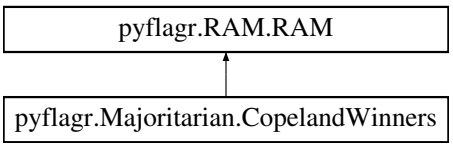
Definition at line 27 of file Majoritarian.py.

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

- PYFLAGR/pyflagr/Majoritarian.py

7.7 pyflagr.Majoritarian.CopelandWinners Class Reference

Inheritance diagram for pyflagr.Majoritarian.CopelandWinners:



Public Member Functions

- def __init__ (self, eval_pts=10)
- def aggregate (self, input_file="", input_df=None, rels_file="", rels_df=None, out_dir=None)

Public Attributes

- output_dir

Additional Inherited Members

7.7.1 Detailed Description

Definition at line 53 of file Majoritarian.py.

7.7.2 Constructor & Destructor Documentation

7.7.2.1 __init__()

```
def pyflagr.Majoritarian.CopelandWinners.__init__ (
              self,
              eval_pts = 10 )
```

Reimplemented from pyflagr.RAM.RAM.

Definition at line 55 of file Majoritarian.py.

7.7.3 Member Function Documentation

7.7.3.1 aggregate()

```
def pyflagr.Majoritarian.CopelandWinners.aggregate (
              self,
              input_file = "",
              input_df = None,
              rels_file = "",
              rels_df = None,
              out_dir = None )
```

Definition at line 68 of file Majoritarian.py.

7.7.4 Member Data Documentation

7.7.4.1 output_dir

```
pyflagr.Majoritarian.CopelandWinners.output_dir
```

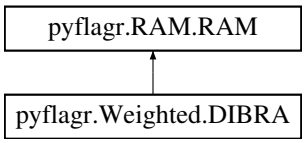
Definition at line 72 of file Majoritarian.py.

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

- PYFLAGR/pyflagr/Majoritarian.py

7.8 pyflagr.Weighted.DIBRA Class Reference

Inheritance diagram for pyflagr.Weighted.DIBRA:



Public Member Functions

- def __init__ (self, eval_pts=10, aggregator='combsum:borda', w_norm='minmax', dist='cosine', prune=False, gamma=1.5, d1=0.4, d2=0.1, tol=0.01, max_iter=50, pref=0.0, veto=0.75, conc=0.0, disc=0.25)
- def aggregate (self, input_file="", input_df=None, rels_file="", rels_df=None, out_dir=None)

Public Attributes

- agg
- weight_norm
- distance_metric
- list_pruning
- g
- d_1
- d_2
- tolerance
- miter
- preference_t
- veto_t
- concordance_t
- discordance_t
- output_dir

Static Public Attributes

- int agg = 5100
- int weight_norm = 2
- int distance_metric = 3
- bool list_pruning = False,
- float g = 1.2,
- float d_1 = 0.4
- float d_2 = 0.1
- float tolerance = 0.01
- int miter = 50
- float preference_t = 0.0
- float veto_t = 0.75
- float concordance_t = 0.0
- float discordance_t = 0.25

7.8.1 Detailed Description

Definition at line 116 of file Weighted.py.

7.8.2 Constructor & Destructor Documentation

7.8.2.1 __init__()

```
def pyflagr.Weighted.DIBRA.__init__ (
              self,
              eval_pts = 10,
              aggregator = 'combsum:borda',
              w_norm = 'minmax',
              dist = 'cosine',
              prune = False,
              gamma = 1.5,
              d1 = 0.4,
              d2 = 0.1,
              tol = 0.01,
              max_iter = 50,
              pref = 0.0,
              veto = 0.75,
              conc = 0.0,
              disc = 0.25 )
```

Reimplemented from pyflagr.RAM.RAM.

Definition at line 131 of file Weighted.py.

7.8.3 Member Function Documentation

7.8.3.1 aggregate()

```
def pyflagr.Weighted.DIBRA.aggregate (
              self,
              input_file = "",
              input_df = None,
              rels_file = "",
              rels_df = None,
              out_dir = None )
```

Definition at line 219 of file Weighted.py.

7.8.4 Member Data Documentation

7.8.4.1 agg [1/2]

```
int pyflagr.Weighted.DIBRA.agg = 5100  [static]
```

Definition at line 117 of file Weighted.py.

7.8.4.2 agg [2/2]

```
pyflagr.Weighted.DIBRA.agg
```

Definition at line 137 of file Weighted.py.

7.8.4.3 concordance_t [1/2]

```
float pyflagr.Weighted.DIBRA.concordance_t = 0.0  [static]
```

Definition at line 128 of file Weighted.py.

7.8.4.4 concordance_t [2/2]

```
pyflagr.Weighted.DIBRA.concordance_t
```

Definition at line 193 of file Weighted.py.

7.8.4.5 d_1 [1/2]

```
float pyflagr.Weighted.DIBRA.d_1 = 0.4  [static]
```

Definition at line 122 of file Weighted.py.

7.8.4.6 d_1 [2/2]

```
pyflagr.Weighted.DIBRA.d_1
```

Definition at line 187 of file Weighted.py.

7.8.4.7 d_2 [1/2]

```
float pyflagr.Weighted.DIBRA.d_2 = 0.1  [static]
```

Definition at line 123 of file Weighted.py.

7.8.4.8 d_2 [2/2]

```
pyflagr.Weighted.DIBRA.d_2
```

Definition at line 188 of file Weighted.py.

7.8.4.9 discordance_t [1/2]

```
float pyflagr.Weighted.DIBRA.discordance_t = 0.25  [static]
```

Definition at line 129 of file Weighted.py.

7.8.4.10 discordance_t [2/2]

```
pyflagr.Weighted.DIBRA.discordance_t
```

Definition at line 194 of file Weighted.py.

7.8.4.11 distance_metric [1/2]

```
int pyflagr.Weighted.DIBRA.distance_metric = 3  [static]
```

Definition at line 119 of file Weighted.py.

7.8.4.12 distance_metric [2/2]

```
pyflagr.Weighted.DIBRA.distance_metric
```

Definition at line 175 of file Weighted.py.

7.8.4.13 g [1/2]

```
float pyflagr.Weighted.DIBRA.g = 1.2,  [static]
```

Definition at line 121 of file Weighted.py.

7.8.4.14 g [2/2]

```
pyflagr.Weighted.DIBRA.g
```

Definition at line 186 of file Weighted.py.

7.8.4.15 list_pruning [1/2]

```
bool pyflagr.Weighted.DIBRA.list_pruning = False,  [static]
```

Definition at line 120 of file Weighted.py.

7.8.4.16 list_pruning [2/2]

```
pyflagr.Weighted.DIBRA.list_pruning
```

Definition at line 185 of file Weighted.py.

7.8.4.17 miter [1/2]

```
int pyflagr.Weighted.DIBRA.miter = 50  [static]
```

Definition at line 125 of file Weighted.py.

7.8.4.18 miter [2/2]

```
pyflagr.Weighted.DIBRA.miter
```

Definition at line 190 of file Weighted.py.

7.8.4.19 output_dir

```
pyflagr.Weighted.DIBRA.output_dir
```

Definition at line 223 of file Weighted.py.

7.8.4.20 preference_t [1/2]

```
float pyflagr.Weighted.DIBRA.preference_t = 0.0  [static]
```

Definition at line 126 of file Weighted.py.

7.8.4.21 preference_t [2/2]

```
pyflagr.Weighted.DIBRA.preference_t
```

Definition at line 191 of file Weighted.py.

7.8.4.22 tolerance [1/2]

```
float pyflagr.Weighted.DIBRA.tolerance = 0.01  [static]
```

Definition at line 124 of file Weighted.py.

7.8.4.23 tolerance [2/2]

```
pyflagr.Weighted.DIBRA.tolerance
```

Definition at line 189 of file Weighted.py.

7.8.4.24 veto_t [1/2]

```
float pyflagr.Weighted.DIBRA.veto_t = 0.75  [static]
```

Definition at line 127 of file Weighted.py.

7.8.4.25 veto_t [2/2]

```
pyflagr.Weighted.DIBRA.veto_t
```

Definition at line 192 of file Weighted.py.

7.8.4.26 weight_norm [1/2]

```
int pyflagr.Weighted.DIBRA.weight_norm = 2  [static]
```

Definition at line 118 of file Weighted.py.

7.8.4.27 weight_norm [2/2]

```
pyflagr.Weighted.DIBRA.weight_norm
```

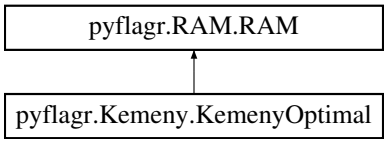
Definition at line 166 of file Weighted.py.

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

- PYFLAGR/pyflagr/Weighted.py

7.9 pyflagr.Kemeny.KemenyOptimal Class Reference

Inheritance diagram for pyflagr.Kemeny.KemenyOptimal:



Public Member Functions

- def __init__ (self, eval_pts=10)
- def aggregate (self, input_file="", input_df=None, rels_file="", rels_df=None, out_dir=None)

Public Attributes

- output_dir

Additional Inherited Members

7.9.1 Detailed Description

Definition at line 8 of file Kemeny.py.

7.9.2 Constructor & Destructor Documentation

7.9.2.1 __init__()

```
def pyflagr.Kemeny.KemenyOptimal.__init__ (
              self,
              eval_pts = 10 )
```

Reimplemented from pyflagr.RAM.RAM.

Definition at line 10 of file Kemeny.py.

7.9.3 Member Function Documentation

7.9.3.1 aggregate()

```
def pyflagr.Kemeny.KemenyOptimal.aggregate (
              self,
              input_file = "",
              input_df = None,
              rels_file = "",
              rels_df = None,
              out_dir = None )
```

Definition at line 23 of file Kemeny.py.

7.9.4 Member Data Documentation

7.9.4.1 output_dir

```
pyflagr.Kemeny.KemenyOptimal.output_dir
```

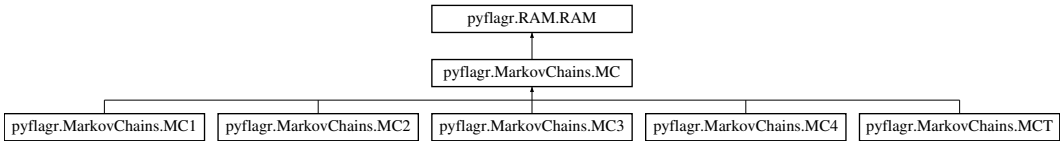
Definition at line 27 of file Kemeny.py.

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

- PYFLAGR/pyflagr/Kemeny.py

7.10 pyflagr.MarkovChains.MC Class Reference

Inheritance diagram for pyflagr.MarkovChains.MC:



Public Member Functions

- def __init__ (self, eval_pts, ergodic_number=def_ergodic_number, max_iterations=def_max_iterations, chain=804)
- def aggregate (self, input_file="", input_df=None, rels_file="", rels_df=None, out_dir=None)

Public Attributes

- chain_type
- output_dir

Static Public Attributes

- erg_num = def_ergodic_number
- niter = def_max_iterations
- int chain_type = 804

7.10.1 Detailed Description

Definition at line 11 of file MarkovChains.py.

7.10.2 Constructor & Destructor Documentation

7.10.2.1 __init__()

```
def pyflagr.MarkovChains.MC.__init__ (
              self,
              eval_pts,
              ergodic_number = def_ergodic_number,
              max_iterations = def_max_iterations,
              chain = 804 )
```

Reimplemented from pyflagr.RAM.RAM.

Reimplemented in pyflagr.MarkovChains.MC1, pyflagr.MarkovChains.MC2, pyflagr.MarkovChains.MC3, pyflagr.MarkovChains.MC4, and pyflagr.MarkovChains.MCT.

Definition at line 16 of file MarkovChains.py.

7.10.3 Member Function Documentation

7.10.3.1 aggregate()

```
def pyflagr.MarkovChains.MC.aggregate (
              self,
              input_file = "",
              input_df = None,
              rels_file = "",
              rels_df = None,
              out_dir = None )
```

Definition at line 37 of file MarkovChains.py.

7.10.4 Member Data Documentation

7.10.4.1 chain_type [1/2]

```
int pyflagr.MarkovChains.MC.chain_type = 804  [static]
```

Definition at line 14 of file MarkovChains.py.

7.10.4.2 chain_type [2/2]

```
pyflagr.MarkovChains.MC.chain_type
```

Definition at line 21 of file MarkovChains.py.

7.10.4.3 erg_num

```
pyflagr.MarkovChains.MC.erg_num = def_ergodic_number  [static]
```

Definition at line 12 of file MarkovChains.py.

7.10.4.4 niter

```
pyflagr.MarkovChains.MC.niter = def_max_iterations  [static]
```

Definition at line 13 of file MarkovChains.py.

7.10.4.5 output_dir

```
pyflagr.MarkovChains.MC.output_dir
```

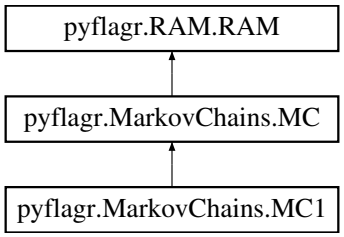
Definition at line 41 of file MarkovChains.py.

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

- PYFLAGR/pyflagr/MarkovChains.py

7.11 pyflagr.MarkovChains.MC1 Class Reference

Inheritance diagram for pyflagr.MarkovChains.MC1:



Public Member Functions

- def __init__ (self, eval_pts, ergodic_number=def_ergodic_number, max_iterations=def_max_iterations)

Additional Inherited Members

7.11.1 Detailed Description

Definition at line 71 of file MarkovChains.py.

7.11.2 Constructor & Destructor Documentation

7.11.2.1 __init__()

```
def pyflagr.MarkovChains.MC1.__init__ (
              self,
              eval_pts,
              ergodic_number = def_ergodic_number,
              max_iterations = def_max_iterations )
```

Reimplemented from pyflagr.MarkovChains.MC.

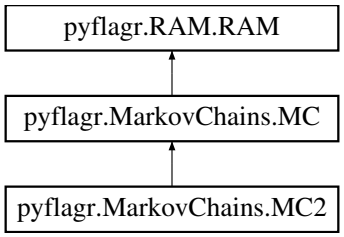
Definition at line 72 of file MarkovChains.py.

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

- PYFLAGR/pyflagr/MarkovChains.py

7.12 pyflagr.MarkovChains.MC2 Class Reference

Inheritance diagram for pyflagr.MarkovChains.MC2:



Public Member Functions

- def __init__ (self, eval_pts, ergodic_number=def_ergodic_number, max_iterations=def_max_iterations)

Additional Inherited Members

7.12.1 Detailed Description

Definition at line 77 of file MarkovChains.py.

7.12.2 Constructor & Destructor Documentation

7.12.2.1 __init__()

```
def pyflagr.MarkovChains.MC2.__init__ (
              self,
              eval_pts,
              ergodic_number = def_ergodic_number,
              max_iterations = def_max_iterations )
```

Reimplemented from pyflagr.MarkovChains.MC.

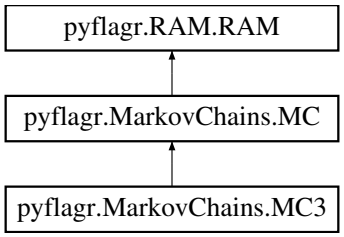
Definition at line 78 of file MarkovChains.py.

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

- PYFLAGR/pyflagr/MarkovChains.py

7.13 pyflagr.MarkovChains.MC3 Class Reference

Inheritance diagram for pyflagr.MarkovChains.MC3:



Public Member Functions

- def __init__ (self, eval_pts, ergodic_number=def_ergodic_number, max_iterations=def_max_iterations)

Additional Inherited Members

7.13.1 Detailed Description

Definition at line 83 of file MarkovChains.py.

7.13.2 Constructor & Destructor Documentation

7.13.2.1 __init__()

```
def pyflagr.MarkovChains.MC3.__init__ (
              self,
              eval_pts,
              ergodic_number = def_ergodic_number,
              max_iterations = def_max_iterations )
```

Reimplemented from pyflagr.MarkovChains.MC.

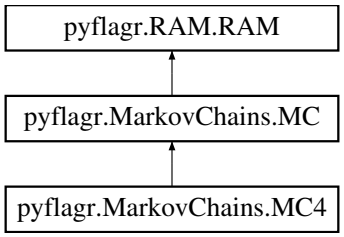
Definition at line 84 of file MarkovChains.py.

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

- PYFLAGR/pyflagr/MarkovChains.py

7.14 pyflagr.MarkovChains.MC4 Class Reference

Inheritance diagram for pyflagr.MarkovChains.MC4:



Public Member Functions

- def __init__ (self, eval_pts, ergodic_number=def_ergodic_number, max_iterations=def_max_iterations)

Additional Inherited Members

7.14.1 Detailed Description

Definition at line 89 of file MarkovChains.py.

7.14.2 Constructor & Destructor Documentation

7.14.2.1 __init__()

```
def pyflagr.MarkovChains.MC4.__init__ (
              self,
              eval_pts,
              ergodic_number = def_ergodic_number,
              max_iterations = def_max_iterations )
```

Reimplemented from pyflagr.MarkovChains.MC.

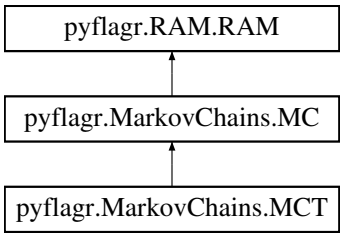
Definition at line 90 of file MarkovChains.py.

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

- PYFLAGR/pyflagr/MarkovChains.py

7.15 pyflagr.MarkovChains.MCT Class Reference

Inheritance diagram for pyflagr.MarkovChains.MCT:



Public Member Functions

- def __init__ (self, eval_pts, ergodic_number=def_ergodic_number, max_iterations=def_max_iterations)

Additional Inherited Members

7.15.1 Detailed Description

Definition at line 95 of file MarkovChains.py.

7.15.2 Constructor & Destructor Documentation

7.15.2.1 __init__()

```
def pyflagr.MarkovChains.MCT.__init__ (
              self,
              eval_pts,
              ergodic_number = def_ergodic_number,
              max_iterations = def_max_iterations )
```

Reimplemented from pyflagr.MarkovChains.MC.

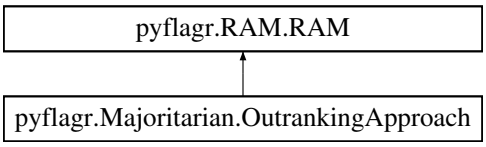
Definition at line 96 of file MarkovChains.py.

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

- PYFLAGR/pyflagr/MarkovChains.py

7.16 pyflagr.Majoritarian.OutrankingApproach Class Reference

Inheritance diagram for pyflagr.Majoritarian.OutrankingApproach:



Public Member Functions

- def __init__ (self, eval_pts=10, preference=0.0, veto=0.75, concordance=0.0, discordance=0.25)
- def aggregate (self, input_file="", input_df=None, rels_file="", rels_df=None, out_dir=None)

Public Attributes

- preference_t
- veto_t
- concordance_t
- discordance_t
- output_dir

Static Public Attributes

- float preference_t = 0.0
- float veto_t = 0.0
- float concordance_t = 0.0
- float discordance_t = 0.0

7.16.1 Detailed Description

Definition at line 98 of file Majoritarian.py.

7.16.2 Constructor & Destructor Documentation

7.16.2.1 __init__()

```
def pyflagr.Majoritarian.OutrankingApproach.__init__ (
              self,
              eval_pts = 10,
              preference = 0.0,
              veto = 0.75,
              concordance = 0.0,
              discordance = 0.25 )
```

Reimplemented from pyflagr.RAM.RAM.

Definition at line 104 of file Majoritarian.py.

7.16.3 Member Function Documentation

7.16.3.1 aggregate()

```
def pyflagr.Majoritarian.OutrankingApproach.aggregate (
              self,
              input_file = "",
              input_df = None,
              rels_file = "",
              rels_df = None,
              out_dir = None )
```

Definition at line 126 of file Majoritarian.py.

7.16.4 Member Data Documentation

7.16.4.1 concordance_t [1/2]

```
float pyflagr.Majoritarian.OutrankingApproach.concordance_t = 0.0  [static]
```

Definition at line 101 of file Majoritarian.py.

7.16.4.2 concordance_t [2/2]

```
pyflagr.Majoritarian.OutrankingApproach.concordance_t
```

Definition at line 109 of file Majoritarian.py.

7.16.4.3 discordance_t [1/2]

```
float pyflagr.Majoritarian.OutrankingApproach.discordance_t = 0.0  [static]
```

Definition at line 102 of file Majoritarian.py.

7.16.4.4 discordance_t [2/2]

```
pyflagr.Majoritarian.OutrankingApproach.discordance_t
```

Definition at line 110 of file Majoritarian.py.

7.16.4.5 output_dir

```
pyflagr.Majoritarian.OutrankingApproach.output_dir
```

Definition at line 130 of file Majoritarian.py.

7.16.4.6 preference_t [1/2]

```
float pyflagr.Majoritarian.OutrankingApproach.preference_t = 0.0  [static]
```

Definition at line 99 of file Majoritarian.py.

7.16.4.7 preference_t [2/2]

```
pyflagr.Majoritarian.OutrankingApproach.preference_t
```

Definition at line 107 of file Majoritarian.py.

7.16.4.8 veto_t [1/2]

```
float pyflagr.Majoritarian.OutrankingApproach.veto_t = 0.0  [static]
```

Definition at line 100 of file Majoritarian.py.

7.16.4.9 veto_t [2/2]

```
pyflagr.Majoritarian.OutrankingApproach.veto_t
```

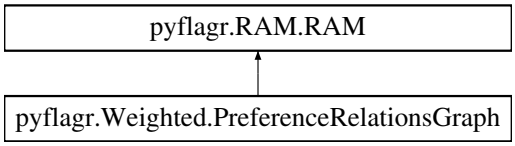
Definition at line 108 of file Majoritarian.py.

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

- PYFLAGR/pyflagr/Majoritarian.py

7.17 pyflagr.Weighted.PreferenceRelationsGraph Class Reference

Inheritance diagram for pyflagr.Weighted.PreferenceRelationsGraph:



Public Member Functions

- def __init__ (self, eval_pts=10, alpha=Alpha, beta=Beta)
- def aggregate (self, input_file="", input_df=None, rels_file="", rels_df=None, out_dir=None)

Public Attributes

- Alpha
- Beta
- output_dir

Static Public Attributes

- float Alpha = 0.1
- float Beta = 0.5

7.17.1 Detailed Description

Definition at line 8 of file Weighted.py.

7.17.2 Constructor & Destructor Documentation

7.17.2.1 __init__()

```
def pyflagr.Weighted.PreferenceRelationsGraph.__init__ (
              self,
              eval_pts = 10,
              alpha = Alpha,
              beta = Beta )
```

Reimplemented from pyflagr.RAM.RAM.

Definition at line 12 of file Weighted.py.

7.17.3 Member Function Documentation

7.17.3.1 aggregate()

```
def pyflagr.Weighted.PreferenceRelationsGraph.aggregate (
              self,
              input_file = "",
              input_df = None,
              rels_file = "",
              rels_df = None,
              out_dir = None )
```

Definition at line 30 of file Weighted.py.

7.17.4 Member Data Documentation

7.17.4.1 Alpha [1/2]

```
float pyflagr.Weighted.PreferenceRelationsGraph.Alpha = 0.1  [static]
```

Definition at line 9 of file Weighted.py.

7.17.4.2 Alpha [2/2]

`pyflagr.Weighted.PreferenceRelationsGraph.Alpha`

Definition at line 15 of file Weighted.py.

7.17.4.3 Beta [1/2]

`float pyflagr.Weighted.PreferenceRelationsGraph.Beta = 0.5 [static]`

Definition at line 10 of file Weighted.py.

7.17.4.4 Beta [2/2]

`pyflagr.Weighted.PreferenceRelationsGraph.Beta`

Definition at line 16 of file Weighted.py.

7.17.4.5 output_dir

`pyflagr.Weighted.PreferenceRelationsGraph.output_dir`

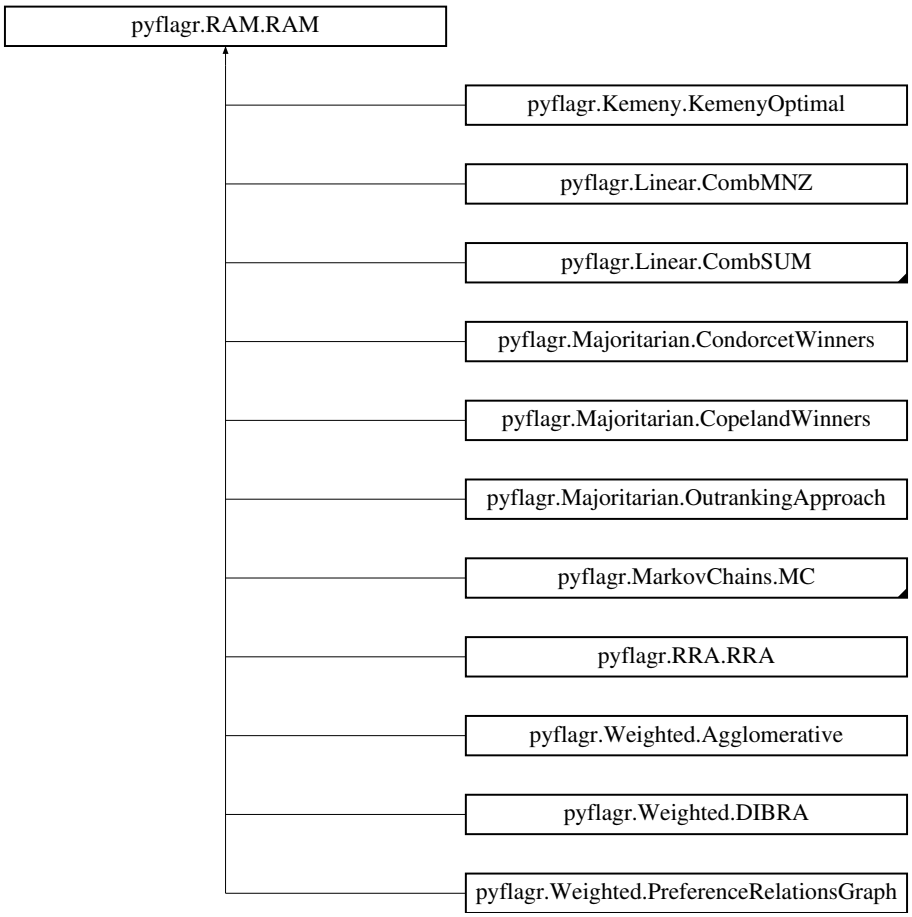
Definition at line 34 of file Weighted.py.

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

- PYFLAGR/pyflagr/Weighted.py

7.18 pyflagr.RAM.RAM Class Reference

Inheritance diagram for pyflagr.RAM.RAM:



Public Member Functions

- def __init__ (self, eval_pts)
- def check_get_input (self, f, df)
- def check_get_rels_input (self, rf, rdf)
- def get_random_string (self, length)
- def get_output (self, od, ran)

Public Attributes

- eval_pts
- flagr_lib
- input_file
- rels_file

Static Public Attributes

- string input_file = ""
- input_df = None
- string rels_file = ""
- rels_df = None
- int eval_pts = 10
- output_dir = tempfile.gettempdir()
- string flagr_lib = ""

7.18.1 Detailed Description

Definition at line 12 of file RAM.py.

7.18.2 Constructor & Destructor Documentation

7.18.2.1 __init__()

```
def pyflagr.RAM.RAM.__init__ (
              self,
              eval_pts )
```

Reimplemented in pyflagr.MarkovChains.MC1, pyflagr.MarkovChains.MC2, pyflagr.MarkovChains.MC3, pyflagr.MarkovChains.MC4, pyflagr.MarkovChains.MCT, pyflagr.MarkovChains.MC, pyflagr.Kemeny.KemenyOptimal, pyflagr.Linear.BordaCount, pyflagr.Linear.SimpleBordaCount, pyflagr.Majoritarian.CondorcetWinners, pyflagr.Majoritarian.CopelandWinners, pyflagr.Weighted.DIBRA, pyflagr.Weighted.PreferenceRelationsGraph, pyflagr.Weighted.Agglomerative, pyflagr.RRA.RRA, pyflagr.Majoritarian.OutrankingApproach, pyflagr.Linear.CombSUM, and pyflagr.Linear.CombMNZ.

Definition at line 21 of file RAM.py.

7.18.3 Member Function Documentation

7.18.3.1 check_get_input()

```
def pyflagr.RAM.RAM.check_get_input (
              self,
              f,
              df )
```

Definition at line 44 of file RAM.py.

7.18.3.2 check_get_rels_input()

```
def pyflagr.RAM.RAM.check_get_rels_input (
              self,
              rf,
              rdf )
```

Definition at line 63 of file RAM.py.

7.18.3.3 get_output()

```
def pyflagr.RAM.RAM.get_output (
              self,
              od,
              ran )
```

Definition at line 86 of file RAM.py.

7.18.3.4 get_random_string()

```
def pyflagr.RAM.RAM.get_random_string (
              self,
              length )
```

Definition at line 78 of file RAM.py.

7.18.4 Member Data Documentation

7.18.4.1 eval_pts [1/2]

```
int pyflagr.RAM.RAM.eval_pts = 10  [static]
```

Definition at line 17 of file RAM.py.

7.18.4.2 eval_pts [2/2]

```
pyflagr.RAM.RAM.eval_pts
```

Definition at line 22 of file RAM.py.

7.18.4.3 flagr_lib [1/2]

```
string pyflagr.RAM.RAM.flagr_lib = ""  [static]
```

Definition at line 19 of file RAM.py.

7.18.4.4 flagr_lib [2/2]

```
pyflagr.RAM.RAM.flagr_lib
```

Definition at line 24 of file RAM.py.

7.18.4.5 input_df

```
pyflagr.RAM.RAM.input_df = None  [static]
```

Definition at line 14 of file RAM.py.

7.18.4.6 input_file [1/2]

```
string pyflagr.RAM.RAM.input_file = ""  [static]
```

Definition at line 13 of file RAM.py.

7.18.4.7 input_file [2/2]

```
pyflagr.RAM.RAM.input_file
```

Definition at line 47 of file RAM.py.

7.18.4.8 output_dir

```
pyflagr.RAM.RAM.output_dir = tempfile.gettempdir()  [static]
```

Definition at line 18 of file RAM.py.

7.18.4.9 rels_df

```
pyflagr.RAM.RAM.rels_df = None  [static]
```

Definition at line 16 of file RAM.py.

7.18.4.10 rels_file [1/2]

```
string pyflagr.RAM.RAM.rels_file = ""  [static]
```

Definition at line 15 of file RAM.py.

7.18.4.11 rels_file [2/2]

```
pyflagr.RAM.RAM.rels_file
```

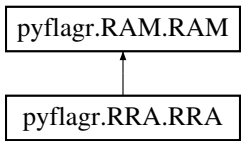
Definition at line 66 of file RAM.py.

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

- PYFLAGR/pyflagr/RAM.py

7.19 pyflagr.RRA.RRA Class Reference

Inheritance diagram for pyflagr.RRA.RRA:



Public Member Functions

- def __init__ (self, eval_pts=10, exact=False)
- def aggregate (self, input_file="", input_df=None, rels_file="", rels_df=None, out_dir=None)

Public Attributes

- exact
- output_dir

Static Public Attributes

- bool exact = False

7.19.1 Detailed Description

Definition at line 8 of file RRA.py.

7.19.2 Constructor & Destructor Documentation

7.19.2.1 __init__()

```
def pyflagr.RRA.RRA.__init__ (
              self,
              eval_pts = 10,
              exact = False )
```

Reimplemented from pyflagr.RAM.RAM.

Definition at line 11 of file RRA.py.

7.19.3 Member Function Documentation

7.19.3.1 aggregate()

```
def pyflagr.RRA.RRA.aggregate (
              self,
              input_file = "",
              input_df = None,
              rels_file = "",
              rels_df = None,
              out_dir = None )
```

Definition at line 27 of file RRA.py.

7.19.4 Member Data Documentation

7.19.4.1 exact [1/2]

```
bool pyflagr.RRA.RRA.exact = False  [static]
```

Definition at line 9 of file RRA.py.

7.19.4.2 exact [2/2]

```
pyflagr.RRA.RRA.exact
```

Definition at line 14 of file RRA.py.

7.19.4.3 output_dir

```
pyflagr.RRA.RRA.output_dir
```

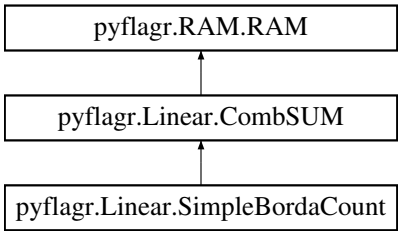
Definition at line 31 of file RRA.py.

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

- PYFLAGR/pyflagr/RRA.py

7.20 pyflagr.Linear.SimpleBordaCount Class Reference

Inheritance diagram for pyflagr.Linear.SimpleBordaCount:



Public Member Functions

- def __init__ (self, eval_pts=10)

Additional Inherited Members

7.20.1 Detailed Description

Definition at line 77 of file Linear.py.

7.20.2 Constructor & Destructor Documentation

7.20.2.1 __init__()

```
def pyflagr.Linear.SimpleBordaCount.__init__ (
              self,
              eval_pts = 10 )
```

Reimplemented from pyflagr.Linear.CombSUM.

Definition at line 78 of file Linear.py.

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

- PYFLAGR/pyflagr/Linear.py

Chapter 8

File Documentation

8.1 PYFLAGR/pyflagr/__init__.py File Reference

Namespaces

- namespace pyflagr

8.2 __init__.py

Go to the documentation of this file.

```
00001 #!/usr/bin/env python
00002 # -*- coding:  utf-8 -*-
00003
00004 #from .RAM import *
00005 #from .Comb import *
00006 #from .Majoritarian import *
00007 #from .MarkovChains import *
00008 #from .Weighted import *
00009
00010 #__version__ = '1.0.2'
```

8.3 PYFLAGR/pyflagr/Comparator.py File Reference

Classes

- class pyflagr.Comparator.Comparator

Namespaces

- namespace pyflagr
- namespace pyflagr.Comparator

8.4 Comparator.py

Go to the documentation of this file.

```
00001 import pandas as pd
00002 import matplotlib.pyplot as plt
00003
00004
00005 # Comparator class to plot/export the values of multiple performance evaluation metrics
00006 class Comparator:
00007     aggregators = None
00008     results = None
00009     ev_pts = 0
00010
00011     def __init__(self, evaluation_points):
00012         self.aggregators = []
00013         self.results = None
00014         self.ev_ptsev_pts = evaluation_points
00015
00016     # Add an aggregator to the local list of aggregators
00017     def add_aggregator(self, name, obj):
00018         self.aggregators.append((name, obj))
00019
00020     # Sequentially invoke the aggregate method of each aggregator.
00021     # Collect the evaluation results in the local Dataframe named self.results.
00022     def aggregate(self, input_file="", input_df=None, rels_file="", rels_df=None):
00023         if len(input_file) == 0 and input_df is None:
00024             print("Error!  You must provide an input file with the preference lists to be
      aggregated.")
00025             return
00026
00027         if len(rels_file) == 0 and rels_df is None:
00028             print("Error!  You must provide a file with the relevance judgements of the list
      elements.")
00029             return
00030
00031         self.results = pd.DataFrame()
00032         for ram in self.aggregators:
00033             print("Running", ram[0], "...")
00034
00035             df_out, df_eval = ram[1].aggregate(input_file, input_df, rels_file, rels_df)
00036
00037             df_eval['Method'] = ram[0]
00038
00039             self.results = pd.concat([self.results, df_eval])
00040
00041         self.results = self.results.rename(columns={'map':  'Mean Average Precision (MAP)'})
00042
00043     # Create a bar plot for MAP
00044     def plot_average_precision(self, dimensions, show_grid, query='all'):
00045         df_new = self.results.loc[lambda df:  df['q'] == query]
00046
00047         df_new.plot.bar(figsize=dimensions, grid=show_grid, x='Method',
00048                         y='Mean Average Precision (MAP)', fontsize=14, position=1)
00049
00050         plt.xlabel('', fontsize=14)
00051         plt.ylabel('Mean Average Precision', fontsize=14)
00052         plt.legend(prop={'size':  14})
00053         plt.show()
00054
00055     # Create a plot for a specific metric at a given cutoff point.  The cutoff point must be lower
      than self.ev_pts
00056     def plot_metric(self, cutoff, metric, plot_type='bar', dimensions=(10.24, 7.68), show_grid=True,
      query='all'):
00057         assert cutoff <= self.ev_ptsev_pts, 'Cannot plot ' + metric + ' at cutoff point ' +
      str(cutoff) + \
00058                                       '.  The length of the list is ' + self.ev_ptsev_pts + '.'
00059
00060         df_new = self.results.loc[lambda df:  df['q'] == query]
00061         df_new = self.get_df_slice(cutoff, metric, df_new)
00062
00063         if plot_type == 'bar':
00064             df_new.plot(kind=plot_type, fontsize=14, width=0.8, figsize=dimensions, grid=show_grid)
00065         else:
00066             df_new.plot(kind=plot_type, fontsize=14, figsize=dimensions, grid=show_grid)
00067
00068         plt.xlabel('List cutoff point', fontsize=14)
00069         plt.ylabel(metric, fontsize=14)
00070         plt.legend(bbox_to_anchor=(1.0, 1.05), prop={'size':  14})
00071         plt.show()
00072
00073     # Slice the evaluation dataframe by specifying rows (query) and columns (metric)
00074     def get_df_slice(self, cutoff, metric, df):
00075         left_columns = 5
00076         if metric == 'map':
00077             start_col = 4
```

```
00078             end_col = 5
00079         elif metric == 'precision':
00080             start_col = left_columns
00081             end_col = start_col + cutoff
00082         elif metric == 'recall':
00083             start_col = left_columns + self.ev_ptsev_pts
00084             end_col = start_col + cutoff
00085         elif metric == 'dcg':
00086             start_col = left_columns + 2 * self.ev_ptsev_pts
00087             end_col = start_col + cutoff
00088         elif metric == 'ndcg':
00089             start_col = left_columns + 3 * self.ev_ptsev_pts
00090             end_col = start_col + cutoff
00091         else:
00092             print(metric, ":  Not supported metric.  Please use one of the following:\n")
00093             print("\tprecision\n\trecall\n\tdcg\n\tndcg\n")
00094             return
00095 
00096         df_ret = df.iloc[:, start_col:end_col].T
00097         df_ret.columns = df.iloc[:, -1]
00098         return df_ret
00099 
00100     # Slice the evaluation dataframe by specifying rows (query) and columns (metric)
00101     def get_results(self, cutoff, metric='all', query='all'):
00102         df_ret = self.results.loc[lambda df:  df['q'] == query]
00103 
00104         if metric != 'all':
00105             df_x = pd.DataFrame()
00106 
00107             for m in metric:
00108                 df_slice = self.get_df_slice(cutoff, m, df_ret)
00109                 df_x = pd.concat([df_x, df_slice], axis=0)
00110 
00111             df_ret = df_x.T
00112 
00113         return df_ret
00114 
00115     # Convert the evaluation results to a LaTeX tabular
00116     def convert_to_latex(self, cutoff, metric='all', query='all', dec_pts=6):
00117         # return self.get_results(cutoff, metric, query).style.format(precision=dec_pts).to_latex()
00118         return self.get_results(cutoff, metric, query).round(dec_pts).to_latex()
00119 
```

8.5 PYFLAGR/pyflagr/Kemeny.py File Reference

Classes

- class pyflagr.Kemeny.KemenyOptimal

Namespaces

- namespace pyflagr
- namespace pyflagr.Kemeny

8.6 Kemeny.py

Go to the documentation of this file.

```
00001 import os.path
00002 import ctypes
00003 
00004 from pyflagr.RAM import RAM
00005 
00006 
00007 # KEMENY OPTIMAL AGGREGATION METHOD
      ================================================================================
00008 class KemenyOptimal(RAM):
00009 
00010     def __init__(self, eval_pts=10):
00011         RAM.__init__(self, eval_pts)
```

```
00012
00013         self.flagr_libflagr_lib.Kemeny.argtypes = [
00014             ctypes.c_char_p,  # Input data file with the lists to be aggregated
00015             ctypes.c_char_p,  # Input data file with the relevant elements per query - used for
      evaluation
00016             ctypes.c_int,     # Number of evaluation points
00017             ctypes.c_char_p,  # Random string to be embedded into the output file names
00018             ctypes.c_char_p   # The directory where the output files will be written
00019         ]
00020
00021         self.flagr_libflagr_lib.Kemeny.restype = None
00022
00023     def aggregate(self, input_file="", input_df=None, rels_file="", rels_df=None, out_dir=None):
00024         # This is the directory where the output files are written.  If nothing is provided, then the
      preset temp
00025         # directory of the OS is used.  If an invalid path is provided, the aforementioned temp dir is
      used silently.
00026         if out_dir is not None and os.path.isdir(out_dir):
00027             self.output_diroutput_dir = out_dir
00028
00029         status = self.check_get_input(input_file, input_df)
00030         if status != 0:
00031             return
00032
00033         status = self.check_get_rels_input(rels_file, rels_df)
00034         if status != 0:
00035             return
00036
00037         ran_str = self.get_random_string(16)
00038
00039         # Call the exposed Kemeny C function
00040         self.flagr_libflagr_lib.Kemeny(
00041             bytes(self.input_fileinput_file, 'ASCII'),
00042             bytes(self.rels_filerels_file, 'ASCII'),
00043             self.eval_ptseval_pts,
00044             bytes(ran_str, 'ASCII'),
00045             bytes(self.output_diroutput_dir, 'ASCII')
00046         )
00047
00048         df_out, df_eval = self.get_output(self.output_diroutput_dir, ran_str)
00049         return df_out, df_eval
```

8.7 PYFLAGR/pyflagr/Linear.py File Reference

Classes

- class pyflagr.Linear.CombSUM
- class pyflagr.Linear.BordaCount
- class pyflagr.Linear.SimpleBordaCount
- class pyflagr.Linear.CombMNZ

Namespaces

- namespace pyflagr
- namespace pyflagr.Linear

8.8 Linear.py

Go to the documentation of this file.

```
00001 import os.path
00002 import ctypes
00003
00004 from pyflagr.RAM import RAM
00005
00006
00007 # COMB SUM
      ============================================================================================================
00008 class CombSUM(RAM):
```

```
00009     normalization = "borda"
00010
00011     def __init__(self, norm="borda", eval_pts=10):
00012         RAM.__init__(self, eval_pts)
00013
00014         self.normalizationnormalization = norm
00015         self.flagr_libflagr_lib.Linear.argtypes = [
00016             ctypes.c_char_p,  # Input data file with the lists to be aggregated
00017             ctypes.c_char_p,  # Input data file with the relevant elements per query - used for
      evaluation
00018             ctypes.c_int,     # Number of evaluation points
00019             ctypes.c_int,     # Rank/Score normalization method (Rank Aggregation Method)
00020             ctypes.c_char_p,  # Random string to be embedded into the output file names
00021             ctypes.c_char_p]  # The directory where the output files will be written
00022
00023         self.flagr_libflagr_lib.Linear.restype = None
00024
00025     def aggregate(self, input_file="", input_df=None, rels_file="", rels_df=None, out_dir=None):
00026         # This is the directory where the output files are written.  If nothing is provided, then the
      preset temp
00027         # directory of the OS is used.  If an invalid path is provided, the aforementioned temp dir is
      used silently.
00028         if out_dir is not None and os.path.isdir(out_dir):
00029             self.output_diroutput_dir = out_dir
00030
00031         status = self.check_get_input(input_file, input_df)
00032         if status != 0:
00033             return
00034
00035         status = self.check_get_rels_input(rels_file, rels_df)
00036
00037         if status != 0:
00038             return
00039
00040         # Rank/Score normalization
00041         if self.normalizationnormalization == "borda":
00042             ram = 100
00043         elif self.normalizationnormalization == "rank":
00044             ram = 101
00045         elif self.normalizationnormalization == "score":
00046             ram = 102
00047         elif self.normalizationnormalization == "z-score":
00048             ram = 103
00049         elif self.normalizationnormalization == "simple-borda":
00050             ram = 104
00051         else:
00052             ram = 100
00053
00054         ran_str = self.get_random_string(16)
00055
00056         # Call the exposed Linear C function
00057         self.flagr_libflagr_lib.Linear(
00058             bytes(self.input_fileinput_file, 'ASCII'),
00059             bytes(self.rels_filerels_file, 'ASCII'),
00060             self.eval_ptseval_pts,
00061             ram,
00062             bytes(ran_str, 'ASCII'),
00063             bytes(self.output_diroutput_dir, 'ASCII')
00064         )
00065
00066         df_out, df_eval = self.get_output(self.output_diroutput_dir, ran_str)
00067         return df_out, df_eval
00068
00069
00070 # BORDA COUNT IS EQUIVALENT TO COMBSUM WITH BORDA NORMALIZATION
      ====================================================
00071 class BordaCount(CombSUM):
00072     def __init__(self, eval_pts=10):
00073         CombSUM.__init__(self, "borda", eval_pts)
00074
00075
00076 # SIMPLE BORDA COUNT IS EQUIVALENT TO COMBSUM WITH BORDA NORMALIZATION
      ============================================
00077 class SimpleBordaCount(CombSUM):
00078     def __init__(self, eval_pts=10):
00079         CombSUM.__init__(self, "simple-borda", eval_pts)
00080
00081
00082 # COMB MNZ
      ============================================================================================================
00083 class CombMNZ(RAM):
00084     normalization = "borda"
00085
00086     def __init__(self, norm="borda", eval_pts=10):
00087         RAM.__init__(self, eval_pts)
00088
00089         self.normalizationnormalization = norm
```

```
00090         self.flagr_libflagr_lib.Linear.argtypes = [
00091             ctypes.c_char_p,  # Input data file with the lists to be aggregated
00092             ctypes.c_char_p,  # Input data file with the relevant elements per query - used for
      evaluation
00093             ctypes.c_int,     # Number of evaluation points
00094             ctypes.c_int,     # Rank/Score normalization method (Rank Aggregation Method)
00095             ctypes.c_char_p,  # Random string to be embedded into the output file names
00096             ctypes.c_char_p]  # The directory where the output files will be written
00097 
00098         self.flagr_libflagr_lib.Linear.restype = None
00099 
00100     def aggregate(self, input_file="", input_df=None, rels_file="", rels_df=None, out_dir=None):
00101         # This is the directory where the output files are written.  If nothing is provided, then the
      preset temp
00102         # directory of the OS is used.  If an invalid path is provided, the aforementioned temp dir is
      used silently.
00103         if out_dir is not None and os.path.isdir(out_dir):
00104             self.output_diroutput_dir = out_dir
00105 
00106         status = self.check_get_input(input_file, input_df)
00107         if status != 0:
00108             return
00109 
00110         status = self.check_get_rels_input(rels_file, rels_df)
00111         if status != 0:
00112             return
00113 
00114         # Rank/Score normalization
00115         if self.normalizationnormalization == "borda":
00116             ram = 110
00117         elif self.normalizationnormalization == "rank":
00118             ram = 111
00119         elif self.normalizationnormalization == "score":
00120             ram = 112
00121         elif self.normalizationnormalization == "z-score":
00122             ram = 113
00123         elif self.normalizationnormalization == "simple-borda":
00124             ram = 114
00125         else:
00126             ram = 110
00127 
00128         ran_str = self.get_random_string(16)
00129 
00130         # Call the exposed Linear C function
00131         self.flagr_libflagr_lib.Linear(
00132             ctypes.c_char_p(self.input_fileinput_file.encode('ascii')),
00133             ctypes.c_char_p(self.rels_filerels_file.encode('ascii')),
00134             self.eval_ptseval_pts,
00135             ram,
00136             ctypes.c_char_p(ran_str.encode('ascii')),
00137             ctypes.c_char_p(self.output_diroutput_dir.encode('ascii'))
00138             )
00139 
00140         df_out, df_eval = self.get_output(self.output_diroutput_dir, ran_str)
00141 
00142         return df_out, df_eval
```

8.9 PYFLAGR/pyflagr/Majoritarian.py File Reference

Classes

- class pyflagr.Majoritarian.CondorcetWinners
- class pyflagr.Majoritarian.CopelandWinners
- class pyflagr.Majoritarian.OutrankingApproach

Namespaces

- namespace pyflagr
- namespace pyflagr.Majoritarian

8.10 Majoritarian.py

Go to the documentation of this file.

```
00001 import os.path
00002 import ctypes
00003
00004 from pyflagr.RAM import RAM
00005
00006
00007 # CONDORCET WINNERS METHOD
      ============================================================================================
00008 class CondorcetWinners(RAM):
00009
00010     def __init__(self, eval_pts=10):
00011         RAM.__init__(self, eval_pts)
00012
00013         self.flagr_libflagr_lib.Condorcet.argtypes = [
00014             ctypes.c_char_p,  # Input data file with the lists to be aggregated
00015             ctypes.c_char_p,  # Input data file with the relevant elements per query - used for
      evaluation
00016             ctypes.c_int,     # Number of evaluation points
00017             ctypes.c_char_p,  # Random string to be embedded into the output file names
00018             ctypes.c_char_p  # The directory where the output files will be written
00019         ]
00020
00021         self.flagr_libflagr_lib.Condorcet.restype = None
00022
00023     def aggregate(self, input_file="", input_df=None, rels_file="", rels_df=None, out_dir=None):
00024         # This is the directory where the output files are written.  If nothing is provided, then the
      preset temp
00025         # directory of the OS is used.  If an invalid path is provided, the aforementioned temp dir is
      used silently.
00026         if out_dir is not None and os.path.isdir(out_dir):
00027             self.output_diroutput_dir = out_dir
00028
00029         status = self.check_get_input(input_file, input_df)
00030         if status != 0:
00031             return
00032
00033         status = self.check_get_rels_input(rels_file, rels_df)
00034         if status != 0:
00035             return
00036
00037         ran_str = self.get_random_string(16)
00038
00039         # Call the exposed Condorcet C function
00040         self.flagr_libflagr_lib.Condorcet(
00041             bytes(self.input_fileinput_file, 'ASCII'),
00042             bytes(self.rels_filerels_file, 'ASCII'),
00043             self.eval_ptseval_pts,
00044             bytes(ran_str, 'ASCII'),
00045             bytes(self.output_diroutput_dir, 'ASCII')
00046         )
00047
00048         df_out, df_eval = self.get_output(self.output_diroutput_dir, ran_str)
00049         return df_out, df_eval
00050
00051
00052 # COPELAND WINNERS METHOD
      ============================================================================================
00053 class CopelandWinners(RAM):
00054
00055     def __init__(self, eval_pts=10):
00056         RAM.__init__(self, eval_pts)
00057
00058         self.flagr_libflagr_lib.Copeland.argtypes = [
00059             ctypes.c_char_p,  # Input data file with the lists to be aggregated
00060             ctypes.c_char_p,  # Input data file with the relevant elements per query - used for
      evaluation
00061             ctypes.c_int,     # Number of evaluation points
00062             ctypes.c_char_p,  # Random string to be embedded into the output file names
00063             ctypes.c_char_p   # The directory where the output files will be written
00064         ]
00065
00066         self.flagr_libflagr_lib.Copeland.restype = None
00067
00068     def aggregate(self, input_file="", input_df=None, rels_file="", rels_df=None, out_dir=None):
00069         # This is the directory where the output files are written.  If nothing is provided, then the
      preset temp
00070         # directory of the OS is used.  If an invalid path is provided, the aforementioned temp dir is
      used silently.
00071         if out_dir is not None and os.path.isdir(out_dir):
00072             self.output_diroutput_dir = out_dir
00073
00074         status = self.check_get_input(input_file, input_df)
```

```
00075         if status != 0:
00076             return
00077 
00078         status = self.check_get_rels_input(rels_file, rels_df)
00079         if status != 0:
00080             return
00081 
00082         ran_str = self.get_random_string(16)
00083 
00084         # Call the exposed Copeland C function
00085         self.flagr_libflagr_lib.Copeland(
00086             bytes(self.input_fileinput_file, 'ASCII'),
00087             bytes(self.rels_filerels_file, 'ASCII'),
00088             self.eval_ptseval_pts,
00089             bytes(ran_str, 'ASCII'),
00090             bytes(self.output_diroutput_dir, 'ASCII')
00091         )
00092 
00093         df_out, df_eval = self.get_output(self.output_diroutput_dir, ran_str)
00094         return df_out, df_eval
00095 
00096 
00097 # OUTRANKING APPROACH
    ================================================================================================
00098 class OutrankingApproach(RAM):
00099     preference_t = 0.0
00100     veto_t = 0.0
00101     concordance_t = 0.0
00102     discordance_t = 0.0
00103 
00104     def __init__(self, eval_pts=10, preference=0.0, veto=0.75, concordance=0.0, discordance=0.25):
00105         RAM.__init__(self, eval_pts)
00106 
00107         self.preference_tpreference_t = preference
00108         self.veto_tveto_t = veto
00109         self.concordance_tconcordance_t = concordance
00110         self.discordance_tdiscordance_t = discordance
00111 
00112         self.flagr_libflagr_lib.OutrankingApproach.argtypes = [
00113             ctypes.c_char_p,  # Input data file with the lists to be aggregated
00114             ctypes.c_char_p,  # Input data file with the relevant elements per query - used for
    evaluation
00115             ctypes.c_int,     # Number of evaluation points
00116             ctypes.c_char_p,  # Random string to be embedded into the output file names
00117             ctypes.c_char_p,  # The directory where the output files will be written
00118             ctypes.c_float,   # Preference Threshold
00119             ctypes.c_float,   # Veto Threshold
00120             ctypes.c_float,   # Concordance Threshold
00121             ctypes.c_float    # Discordance Threshold
00122         ]
00123 
00124         self.flagr_libflagr_lib.OutrankingApproach.restype = None
00125 
00126     def aggregate(self, input_file="", input_df=None, rels_file="", rels_df=None, out_dir=None):
00127         # This is the directory where the output files are written.  If nothing is provided, then the
    preset temp
00128         # directory of the OS is used.  If an invalid path is provided, the aforementioned temp dir is
    used silently.
00129         if out_dir is not None and os.path.isdir(out_dir):
00130             self.output_diroutput_dir = out_dir
00131 
00132         status = self.check_get_input(input_file, input_df)
00133         if status != 0:
00134             return
00135 
00136         status = self.check_get_rels_input(rels_file, rels_df)
00137         if status != 0:
00138             return
00139 
00140         ran_str = self.get_random_string(16)
00141 
00142         # Call the exposed OutrankingApproach C function
00143         self.flagr_libflagr_lib.OutrankingApproach(
00144             bytes(self.input_fileinput_file, 'ASCII'),
00145             bytes(self.rels_filerels_file, 'ASCII'),
00146             self.eval_ptseval_pts,
00147             bytes(ran_str, 'ASCII'),
00148             bytes(self.output_diroutput_dir, 'ASCII'),
00149             float(self.preference_tpreference_t),
00150             float(self.veto_tveto_t),
00151             float(self.concordance_tconcordance_t),
00152             float(self.discordance_tdiscordance_t)
00153         )
00154 
00155         df_out, df_eval = self.get_output(self.output_diroutput_dir, ran_str)
00156         return df_out, df_eval
```

8.11 PYFLAGR/pyflagr/MarkovChains.py File Reference

Classes

- class pyflagr.MarkovChains.MC
- class pyflagr.MarkovChains.MC1
- class pyflagr.MarkovChains.MC2
- class pyflagr.MarkovChains.MC3
- class pyflagr.MarkovChains.MC4
- class pyflagr.MarkovChains.MCT

Namespaces

- namespace pyflagr
- namespace pyflagr.MarkovChains

Variables

- float pyflagr.MarkovChains.def_ergodic_number = 0.15
- int pyflagr.MarkovChains.def_max_iterations = 100

8.12 MarkovChains.py

Go to the documentation of this file.

```
00001 import os.path
00002 import ctypes
00003
00004 from pyflagr.RAM import RAM
00005
00006 def_ergodic_number = 0.15
00007 def_max_iterations = 100
00008
00009
00010 # MARKOV CHAINS BASE CLASS
      ================================================================================================
00011 class MC(RAM):
00012     erg_num = def_ergodic_number
00013     niter = def_max_iterations
00014     chain_type = 804
00015
00016     def __init__(self, eval_pts, ergodic_number=def_ergodic_number, max_iterations=def_max_iterations,
      chain=804):
00017         RAM.__init__(self, eval_pts)
00018
00019         self.erg_num = ergodic_number
00020         self.niter = max_iterations
00021         self.chain_typechain_type = chain
00022
00023         self.flagr_libflagr_lib.MC.argtypes = [
00024             ctypes.c_char_p,  # Input data file with the lists to be aggregated
00025             ctypes.c_char_p,  # Input data file with the relevant elements per query - used for
      evaluation
00026             ctypes.c_int,     # Number of evaluation points
00027             ctypes.c_int,     # Type of Markov Chain
00028             ctypes.c_char_p,  # Random string to be embedded into the output file names
00029             ctypes.c_char_p,  # The directory where the output files will be written
00030             ctypes.c_float,   # Ergodic number
00031             ctypes.c_float,   # delta parameter
00032             ctypes.c_int      # Maximum number of iterations
00033         ]
00034
00035         self.flagr_libflagr_lib.MC.restype = None
00036
00037     def aggregate(self, input_file="", input_df=None, rels_file="", rels_df=None, out_dir=None):
```

```
00038         # This is the directory where the output files are written.  If nothing is provided, then the
      preset temp
00039         # directory of the OS is used.  If an invalid path is provided, the aforementioned temp dir is
      used silently.
00040         if out_dir is not None and os.path.isdir(out_dir):
00041             self.output_diroutput_dir = out_dir
00042 
00043         status = self.check_get_input(input_file, input_df)
00044         if status != 0:
00045             return
00046 
00047         status = self.check_get_rels_input(rels_file, rels_df)
00048         if status != 0:
00049             return
00050 
00051         ran_str = self.get_random_string(16)
00052 
00053         # Call the exposed Condorcet C function
00054         self.flagr_libflagr_lib.MC(
00055             bytes(self.input_fileinput_file, 'ASCII'),
00056             bytes(self.rels_filerels_file, 'ASCII'),
00057             self.eval_ptseval_pts,
00058             self.chain_typechain_type,
00059             bytes(ran_str, 'ASCII'),
00060             bytes(self.output_diroutput_dir, 'ASCII'),
00061             self.erg_num,
00062             0.0,
00063             self.niter
00064         )
00065 
00066         df_out, df_eval = self.get_output(self.output_diroutput_dir, ran_str)
00067         return df_out, df_eval
00068 
00069 
00070 # MARKOV CHAINS 1 METHOD
      ============================================================================================
00071 class MC1(MC):
00072     def __init__(self, eval_pts, ergodic_number=def_ergodic_number,
      max_iterations=def_max_iterations):
00073         MC.__init__(self, eval_pts, ergodic_number, max_iterations, 801)
00074 
00075 
00076 # MARKOV CHAINS 2 METHOD
      ============================================================================================
00077 class MC2(MC):
00078     def __init__(self, eval_pts, ergodic_number=def_ergodic_number,
      max_iterations=def_max_iterations):
00079         MC.__init__(self, eval_pts, ergodic_number, max_iterations, 802)
00080 
00081 
00082 # MARKOV CHAINS 3 METHOD
      ============================================================================================
00083 class MC3(MC):
00084     def __init__(self, eval_pts, ergodic_number=def_ergodic_number,
      max_iterations=def_max_iterations):
00085         MC.__init__(self, eval_pts, ergodic_number, max_iterations, 803)
00086 
00087 
00088 # MARKOV CHAINS 4 METHOD
      ============================================================================================
00089 class MC4(MC):
00090     def __init__(self, eval_pts, ergodic_number=def_ergodic_number,
      max_iterations=def_max_iterations):
00091         MC.__init__(self, eval_pts, ergodic_number, max_iterations, 804)
00092 
00093 
00094 # MARKOV CHAINS THURSTONE METHOD
      ============================================================================================
00095 class MCT(MC):
00096     def __init__(self, eval_pts, ergodic_number=def_ergodic_number,
      max_iterations=def_max_iterations):
00097         MC.__init__(self, eval_pts, ergodic_number, max_iterations, 805)
```

8.13 PYFLAGR/pyflagr/RAM.py File Reference

Classes

- class pyflagr.RAM.RAM

Namespaces

- namespace pyflagr
- namespace pyflagr.RAM

8.14 RAM.py

Go to the documentation of this file.

```
00001 import os
00002
00003 import ctypes
00004 import random
00005 import string
00006 import tempfile
00007 from sys import platform
00008
00009 import pandas as pd
00010
00011
00012 class RAM:
00013     input_file = ""
00014     input_df = None
00015     rels_file = ""
00016     rels_df = None
00017     eval_pts = 10
00018     output_dir = tempfile.gettempdir()
00019     flagr_lib = ""
00020
00021     def __init__(self, eval_pts):
00022         self.eval_ptseval_pts = eval_pts
00023
00024         self.flagr_libflagr_lib = None
00025
00026         # Import the FLAGR shared library in PyFLAGR
00027         if platform == "linux" or platform == "linux2":
00028             self.flagr_libflagr_lib = ctypes.CDLL(os.path.dirname(os.path.realpath(__file__)) +
      "/flagr.so")
00029
00030         elif platform == "win32":
00031             ”’
00032 os.environ['PATH'] = os.path.dirname(os.path.realpath(__file__)) + os.pathsep + os.environ['PATH']
00033             paths = os.environ['PATH'].split(";")
00034             for path in paths:
00035                 if os.path.isdir(path):
00036                     os.add_dll_directory(path)
00037             ”’
00038 self.flagr_libflagr_lib = ctypes.CDLL(os.path.dirname(os.path.realpath(__file__)) + '/flagr.dll')
00039
00040         elif platform == "darwin":
00041             self.flagr_libflagr_lib = ctypes.CDLL(os.path.dirname(os.path.realpath(__file__)) +
      "/flagr.dylib")
00042
00043     # Check the input file or DataFrame that contains the input lists
00044     def check_get_input(self, f, df):
00045         status = 0
00046         if len(f) > 0:
00047             self.input_fileinput_file = f
00048             if not os.path.isfile(self.input_fileinput_file):
00049                 print("Error!  Input file does not exist")
00050                 status = -1
00051
00052         elif df is not None:
00053             self.input_fileinput_file = tempfile.gettempdir() + "/temp_input.csv"
00054             df.to_csv(self.input_fileinput_file, index=False)
00055
00056         else:
00057             print("Error!  No input data was passed")
00058             status = -1
00059
00060         return status
00061
00062     # Check the input file or DataFrame that contains the Relevant elements for each query
00063     def check_get_rels_input(self, rf, rdf):
00064         status = 0
00065         if len(rf) > 0:
00066             self.rels_filerels_file = rf
00067
00068         elif rdf is not None:
00069             self.rels_filerels_file = tempfile.gettempdir() + "/temp_input_rels.csv"
00070             rdf.to_csv(self.rels_filerels_file, index=False)
```

```
00071
00072         else:
00073             self.rels_filerels_file = ""
00074
00075         return status
00076
00077     # A random bytes' generator - to be used in filenames passed to FLAGR
00078     def get_random_string(self, length):
00079         letters = string.ascii_lowercase
00080         result_str = ''.join(random.choice(letters) for _ in range(length))
00081         return result_str
00082
00083     # Retrieve the output from a PyFLAGR rank aggregation method.  This one reads the output CSV files
      and returns
00084     # two DataFrames.  The first one contains the aggregate lists for each query; the second one
      stores the results
00085     # of the evaluation (provided that a qrels file or DataFrame has been set).
00086     def get_output(self, od, ran):
00087         out_file = od + "/out_" + ran + ".csv"
00088         eval_file = od + "/eval_" + ran + ".csv"
00089
00090         if os.path.isfile(out_file):
00091             df_out = pd.read_csv(out_file, engine='c')
00092             if od == tempfile.gettempdir():
00093                 os.remove(out_file)
00094
00095             if os.path.isfile(eval_file):
00096                 df_eval = pd.read_csv(eval_file)
00097                 if od == tempfile.gettempdir():
00098                     os.remove(eval_file)
00099
00100                 return df_out, df_eval
00101             else:
00102                 df_rel = pd.DataFrame()
00103                 return df_out, df_rel
00104
00105         else:
00106             df_out = pd.DataFrame()
00107             df_rel = pd.DataFrame()
00108             return df_out, df_rel
```

8.15 PYFLAGR/pyflagr/RRA.py File Reference

Classes

- class pyflagr.RRA.RRA

Namespaces

- namespace pyflagr
- namespace pyflagr.RRA

8.16 RRA.py

Go to the documentation of this file.

```
00001 import os.path
00002 import ctypes
00003
00004 from pyflagr.RAM import RAM
00005
00006
00007 # ROBUST RANK AGGREGATION METHOD
      ================================================================================
00008 class RRA(RAM):
00009     exact = False
00010
00011     def __init__(self, eval_pts=10, exact=False):
00012         RAM.__init__(self, eval_pts)
00013
```

```
00014         self.exactexact = exact
00015
00016         self.flagr_libflagr_lib.RobustRA.argtypes = [
00017             ctypes.c_char_p,  # Input data file with the lists to be aggregated
00018             ctypes.c_char_p,  # Input data file with the relevant elements per query - used for
      evaluation
00019             ctypes.c_int,     # Number of evaluation points
00020             ctypes.c_char_p,  # Random string to be embedded into the output file names
00021             ctypes.c_char_p,  # The directory where the output files will be written
00022             ctypes.c_bool     # Correct p-values with Stuart-Ares algorithm
00023         ]
00024
00025         self.flagr_libflagr_lib.RobustRA.restype = None
00026
00027     def aggregate(self, input_file="", input_df=None, rels_file="", rels_df=None, out_dir=None):
00028         # This is the directory where the output files are written.  If nothing is provided, then the
      preset temp
00029         # directory of the OS is used.  If an invalid path is provided, the aforementioned temp dir is
      used silently.
00030         if out_dir is not None and os.path.isdir(out_dir):
00031             self.output_diroutput_dir = out_dir
00032
00033         status = self.check_get_input(input_file, input_df)
00034         if status != 0:
00035             return
00036
00037         status = self.check_get_rels_input(rels_file, rels_df)
00038         if status != 0:
00039             return
00040
00041         ran_str = self.get_random_string(16)
00042
00043         # Call the exposed RobustRA C function
00044         self.flagr_libflagr_lib.RobustRA(
00045             bytes(self.input_fileinput_file, 'ASCII'),
00046             bytes(self.rels_filerels_file, 'ASCII'),
00047             self.eval_ptseval_pts,
00048             bytes(ran_str, 'ASCII'),
00049             bytes(self.output_diroutput_dir, 'ASCII'),
00050             self.exactexact
00051         )
00052
00053         df_out, df_eval = self.get_output(self.output_diroutput_dir, ran_str)
00054         return df_out, df_eval
```

8.17 PYFLAGR/pyflagr/test_local.py File Reference

Namespaces

- namespace pyflagr
- namespace pyflagr.test_local

Variables

- string pyflagr.test_local.base_path = ''
- string pyflagr.test_local.lists = base_path + 'MOSO.csv'
- string pyflagr.test_local.qrels = base_path + 'MOSO_qrels.csv'
- pyflagr.test_local.input_dataframe = pd.read_csv(lists)
- pyflagr.test_local.rels_dataframe = pd.read_csv(qrels)
- int pyflagr.test_local.EV_PTS = 10
- pyflagr.test_local.cmp = Comparator.Comparator(EV_PTS)
- pyflagr.test_local.input_file
- pyflagr.test_local.rels_file

8.18 test_local.py

Go to the documentation of this file.

```
00001
00003 import os.path
00004 import sys
00005
00006 # Comment the following line to execute the code from the installed library.  Otherwise, Python
      executes the local files.
00007 sys.path.insert(1, os.path.dirname(sys.path[0]))
00008
00009 import Linear
00010 import Majoritarian
00011 import MarkovChains
00012 # import Kemeny
00013 import RRA
00014 import Weighted
00015 import Comparator
00016
00017 from sys import platform
00018
00019 import pandas as pd
00020
00021 if __name__ == '__main__':
00022     base_path = ''
00023     if platform == "linux" or platform == "linux2":
00024         base_path = '/media/leo/B65266EC5266B133/phd_Research/08 - Datasets/TREC/Synthetic/'
00025     elif platform == "win32":
00026         base_path = 'D:/phd_Research/08 - Datasets/TREC/Synthetic/'
00027     else:
00028         exit(1)
00029
00030     lists = base_path + 'MOSO.csv'
00031     qrels = base_path + 'MOSO_qrels.csv'
00032
00033     input_dataframe = pd.read_csv(lists)
00034     rels_dataframe = pd.read_csv(qrels)
00035
00036     # method = Linear.BordaCount(eval_pts=20)
00037     # method = Linear.SimpleBordaCount(eval_pts=20)
00038     # method = Linear.CombSUM(eval_pts=20, norm="rank")
00039     # method = Linear.CombSUM(eval_pts=20, norm="score")
00040     # method = Linear.CombMNZ(eval_pts=20, norm="borda")
00041     # method = Linear.CombMNZ(eval_pts=20, norm="simple-borda")
00042     # method = Majoritarian.CondorcetWinners(eval_pts=20)
00043     # method = Majoritarian.CopelandWinners(eval_pts=20)
00044     # method = Majoritarian.OutrankingApproach(eval_pts=20)
00045     # method = RRA.RRA(eval_pts=20, exact=False)
00046     # method = RRA.RRA(eval_pts=20, exact=True)
00047     # method = Weighted.PreferenceRelationsGraph(eval_pts=20)
00048     # method = Weighted.Agglomerative(eval_pts=20)
00049     # method = Weighted.DIBRA(eval_pts=20, aggregator="combmnz:simple-borda")
00050     # method = Weighted.DIBRA(eval_pts=20, gamma=1.5, prune=True, d1=0.4, d2=0.1)
00051     # method = Weighted.DIBRA(eval_pts=10, aggregator="condorcet", w_norm="minmax",
00052     # dist="cosine", prune=False, gamma=1.5, d1=0.4, d2=0.1, max_iter=50)
00053     # method = MarkovChains.MC1(eval_pts=20, ergodic_number=0.15)
00054     # method = MarkovChains.MC2(eval_pts=20, ergodic_number=0.15)
00055     # method = MarkovChains.MC3(eval_pts=20, ergodic_number=0.15)
00056     # method = MarkovChains.MC4(eval_pts=20, ergodic_number=0.15)
00057     # method = MarkovChains.MCT(eval_pts=20, ergodic_number=0.15)
00058
00059     # df_out, df_eval = method.aggregate(input_file=lists, rels_file=qrels,
      out_dir='/home/leo/Documents')
00060     # print(df_eval)
00061
00062     EV_PTS = 10
00063
00064     cmp = Comparator.Comparator(EV_PTS)
00065     cmp.add_aggregator("CombSUM-Rank", Linear.CombSUM(norm='rank', eval_pts=EV_PTS))
00066     cmp.add_aggregator("CombSUM-Borda", Linear.CombSUM(norm='borda', eval_pts=EV_PTS))
00067     cmp.add_aggregator("CombSUM-Score", Linear.CombSUM(norm='score', eval_pts=EV_PTS))
00068     cmp.add_aggregator("CombMNZ-Rank", Linear.CombMNZ(norm='rank', eval_pts=EV_PTS))
00069     cmp.add_aggregator("CombMNZ-Borda", Linear.CombMNZ(norm='borda', eval_pts=EV_PTS))
00070     cmp.add_aggregator("CombMNZ-Score", Linear.CombMNZ(norm='score', eval_pts=EV_PTS))
00071     cmp.add_aggregator("Condorcet", Majoritarian.CondorcetWinners(eval_pts=EV_PTS))
00072     cmp.add_aggregator("Copeland", Majoritarian.CopelandWinners(eval_pts=EV_PTS))
00073     cmp.add_aggregator("Outranking Approach", Majoritarian.OutrankingApproach(preference=0, veto=0.75,
      concordance=0,
00074                                                                               discordance=0.25,
      eval_pts=EV_PTS))
00075     cmp.add_aggregator("MC1", MarkovChains.MC1(max_iterations=50, ergodic_number=0.15,
      eval_pts=EV_PTS))
00076     cmp.add_aggregator("MC2", MarkovChains.MC2(max_iterations=50, ergodic_number=0.15,
      eval_pts=EV_PTS))
00077     cmp.add_aggregator("MC3", MarkovChains.MC3(max_iterations=50, ergodic_number=0.15,
      eval_pts=EV_PTS))
```

```
00078     cmp.add_aggregator("MC4", MarkovChains.MC4(max_iterations=50, ergodic_number=0.15,
      eval_pts=EV_PTS))
00079     cmp.add_aggregator("MCT", MarkovChains.MCT(max_iterations=50, ergodic_number=0.15,
      eval_pts=EV_PTS))
00080     cmp.add_aggregator("RRA-Exact", RRA.RRA(exact=True, eval_pts=EV_PTS))
00081     cmp.add_aggregator("RRA", RRA.RRA(exact=False, eval_pts=EV_PTS))
00082     cmp.add_aggregator("PrefRel", Weighted.PreferenceRelationsGraph(alpha=0.1, beta=0.5,
      eval_pts=EV_PTS))
00083     # cmp.add_aggregator("Agglomerative", Weighted.Agglomerative(c1=0.1, c2=0.2, eval_pts=EV_PTS))
00084     cmp.add_aggregator("DIBRA", Weighted.DIBRA(aggregator='combsum:borda', gamma=1.2, prune=False,
      w_norm='minmax',
00085                                                eval_pts=EV_PTS))
00086     cmp.add_aggregator("DIBRA-Prune", Weighted.DIBRA(aggregator='combsum:borda', gamma=1.2,
      prune=True, w_norm='minmax',
00087                                                      d1=0.3, d2=0.05, eval_pts=EV_PTS))
00088
00089     cmp.aggregate(input_file=lists, rels_file=qrels)
00090     # cmp.plot_average_precision((10.24, 7.68), True, query='all')
00091     # cmp.plot_metric(EV_PTS, metric='ndcg', plot_type='bar', dimensions=(10.24, 7.68),
      show_grid=True, query='all')
00092     # print(cmp.get_results(cutoff=3, metric=["dcg", "ndcg"], query='Topic 1'))
00093     print(cmp.convert_to_latex(dec_pts=4, cutoff=EV_PTS, metric=["map", "precision", "ndcg"],
      query='all'))
```

8.19 PYFLAGR/pyflagr/Weighted.py File Reference

Classes

- class pyflagr.Weighted.PreferenceRelationsGraph
- class pyflagr.Weighted.Agglomerative
- class pyflagr.Weighted.DIBRA

Namespaces

- namespace pyflagr
- namespace pyflagr.Weighted

8.20 Weighted.py

Go to the documentation of this file.

```
00001 import os.path
00002 import ctypes
00003
00004 from pyflagr.RAM import RAM
00005
00006
00007 # PREFERENCE RELATIONS METHOD
      ============================================================================================
00008 class PreferenceRelationsGraph(RAM):
00009     Alpha = 0.1
00010     Beta = 0.5
00011
00012     def __init__(self, eval_pts=10, alpha=Alpha, beta=Beta):
00013         RAM.__init__(self, eval_pts)
00014
00015         self.AlphaAlpha = alpha
00016         self.BetaBeta = beta
00017
00018         self.flagr_libflagr_lib.PrefRel.argtypes = [
00019             ctypes.c_char_p,  # Input data file with the lists to be aggregated
00020             ctypes.c_char_p,  # Input data file with the relevant elements per query - used for
      evaluation
00021             ctypes.c_int,     # Number of evaluation points
00022             ctypes.c_char_p,  # Random string to be embedded into the output file names
00023             ctypes.c_char_p,  # The directory where the output files will be written
00024             ctypes.c_float,   # alpha parameter
00025             ctypes.c_float    # beta parameter
00026         ]
```

```
00027
00028         self.flagr_libflagr_lib.PrefRel.restype = None
00029
00030     def aggregate(self, input_file="", input_df=None, rels_file="", rels_df=None, out_dir=None):
00031         # This is the directory where the output files are written.  If nothing is provided, then the
      preset temp
00032         # directory of the OS is used.  If an invalid path is provided, the aforementioned temp dir is
      used silently.
00033         if out_dir is not None and os.path.isdir(out_dir):
00034             self.output_diroutput_dir = out_dir
00035
00036         status = self.check_get_input(input_file, input_df)
00037         if status != 0:
00038             return
00039
00040         status = self.check_get_rels_input(rels_file, rels_df)
00041         if status != 0:
00042             return
00043
00044         ran_str = self.get_random_string(16)
00045
00046         # Call the exposed PrefRel C function
00047         self.flagr_libflagr_lib.PrefRel(
00048             bytes(self.input_fileinput_file, 'ASCII'),
00049             bytes(self.rels_filerels_file, 'ASCII'),
00050             self.eval_ptseval_pts,
00051             bytes(ran_str, 'ASCII'),
00052             bytes(self.output_diroutput_dir, 'ASCII'),
00053             self.AlphaAlpha,
00054             self.BetaBeta
00055         )
00056
00057         df_out, df_eval = self.get_output(self.output_diroutput_dir, ran_str)
00058         return df_out, df_eval
00059
00060
00061 # AGGLOMERATIVE METHOD
      ============================================================================================
00062 class Agglomerative(RAM):
00063     C1 = 0.1
00064     C2 = 0.5
00065
00066     def __init__(self, eval_pts=10, c1=C1, c2=C2):
00067         RAM.__init__(self, eval_pts)
00068
00069         self.C1C1 = c1
00070         self.C2C2 = c2
00071
00072         self.flagr_libflagr_lib.Agglomerative.argtypes = [
00073             ctypes.c_char_p,  # Input data file with the lists to be aggregated
00074             ctypes.c_char_p,  # Input data file with the relevant elements per query - used for
      evaluation
00075             ctypes.c_int,     # Number of evaluation points
00076             ctypes.c_char_p,  # Random string to be embedded into the output file names
00077             ctypes.c_char_p,  # The directory where the output files will be written
00078             ctypes.c_float,   # c1 parameter
00079             ctypes.c_float    # c2 parameter
00080         ]
00081
00082         self.flagr_libflagr_lib.Agglomerative.restype = None
00083
00084     def aggregate(self, input_file="", input_df=None, rels_file="", rels_df=None, out_dir=None):
00085         # This is the directory where the output files are written.  If nothing is provided, then the
      preset temp
00086         # directory of the OS is used.  If an invalid path is provided, the aforementioned temp dir is
      used silently.
00087         if out_dir is not None and os.path.isdir(out_dir):
00088             self.output_diroutput_dir = out_dir
00089
00090         status = self.check_get_input(input_file, input_df)
00091         if status != 0:
00092             return
00093
00094         status = self.check_get_rels_input(rels_file, rels_df)
00095         if status != 0:
00096             return
00097
00098         ran_str = self.get_random_string(16)
00099
00100         # Call the exposed Agglomerative C function
00101         self.flagr_libflagr_lib.Agglomerative(
00102             bytes(self.input_fileinput_file, 'ASCII'),
00103             bytes(self.rels_filerels_file, 'ASCII'),
00104             self.eval_ptseval_pts,
00105             bytes(ran_str, 'ASCII'),
00106             bytes(self.output_diroutput_dir, 'ASCII'),
00107             self.C1C1,
```

```
00108             self.C2C2
00109         )
00110
00111         df_out, df_eval = self.get_output(self.output_diroutput_dir, ran_str)
00112         return df_out, df_eval
00113
00114
00115 # DIBRA METHOD : DISTANCE-BASED ITERATIVE
      ============================================================================
00116 class DIBRA(RAM):
00117     agg = 5100
00118     weight_norm = 2
00119     distance_metric = 3
00120     list_pruning = False,
00121     g = 1.2,
00122     d_1 = 0.4
00123     d_2 = 0.1
00124     tolerance = 0.01
00125     miter = 50
00126     preference_t = 0.0
00127     veto_t = 0.75
00128     concordance_t = 0.0
00129     discordance_t = 0.25
00130
00131     def __init__(self, eval_pts=10, aggregator='combsum:borda', w_norm='minmax', dist='cosine',
      prune=False,
00132 gamma=1.5, d1=0.4, d2=0.1, tol=0.01, max_iter=50, pref=0.0, veto=0.75, conc=0.0, disc=0.25):
00133
00134 RAM.__init__(self, eval_pts)
00135
00136 # Set the aggregator code according to the selected aggregation method
00137         self.aggagg = 5100
00138         if aggregator == "combsum:borda":
00139             self.aggagg = 5100
00140         elif aggregator == "combsum:rank":
00141             self.aggagg = 5101
00142         elif aggregator == "combsum:score":
00143             self.aggagg = 5102
00144         elif aggregator == "combsum:z-score":
00145             self.aggagg = 5103
00146         elif aggregator == "combsum:simple-borda":
00147             self.aggagg = 5104
00148         elif aggregator == "combmnz:borda":
00149             self.aggagg = 5110
00150         elif aggregator == "combmnz:rank":
00151             self.aggagg = 5111
00152         elif aggregator == "combmnz:score":
00153             self.aggagg = 5112
00154         elif aggregator == "combmnz:z-score":
00155             self.aggagg = 5113
00156         elif aggregator == "combmnz:simple-borda":
00157             self.aggagg = 5114
00158         elif aggregator == "condorcet":
00159             self.aggagg = 5200
00160         elif aggregator == "copeland":
00161             self.aggagg = 5201
00162         elif aggregator == "outrank":
00163             self.aggagg = 5300
00164
00165         # Set the voter weights normalization method
00166         self.weight_normweight_norm = 2
00167         if w_norm == "none":
00168             self.weight_normweight_norm = 1
00169         elif w_norm == "minmax":
00170             self.weight_normweight_norm = 2
00171         elif w_norm == "z":
00172             self.weight_normweight_norm = 3
00173
00174         # Set the list distance metric
00175         self.distance_metricdistance_metric = 3
00176         if dist == "rho":
00177             self.distance_metricdistance_metric = 1
00178         elif dist == "cosine":
00179             self.distance_metricdistance_metric = 3
00180         elif dist == "footrule":
00181             self.distance_metricdistance_metric = 4
00182         elif dist == "tau":
00183             self.distance_metricdistance_metric = 5
00184
00185         self.list_pruninglist_pruning = prune
00186         self.gg = gamma
00187         self.d_1d_1 = d1
00188         self.d_2d_2 = d2
00189         self.tolerancetolerance = tol
00190         self.mitermiter = max_iter
00191         self.preference_tpreference_t = pref
00192         self.veto_tveto_t = veto
```

```
00193         self.concordance_tconcordance_t = conc
00194         self.discordance_tdiscordance_t = disc
00195 
00196         self.flagr_libflagr_lib.DIBRA.argtypes = [
00197             ctypes.c_char_p,  # Input data file with the lists to be aggregated
00198             ctypes.c_char_p,  # Input data file with the relevant elements per query - used for
    evaluation
00199             ctypes.c_int,     # Number of evaluation points
00200             ctypes.c_int,     # basic un-weighted rank aggregation method
00201             ctypes.c_char_p,  # Random string to be embedded into the output file names
00202             ctypes.c_char_p,  # The directory where the output files will be written
00203             ctypes.c_int,     # Voter weights normalization
00204             ctypes.c_int,     # List distance function
00205             ctypes.c_bool,    # List pruning algorithm
00206             ctypes.c_float,   # gamma parameter
00207             ctypes.c_float,   # d1 parameter
00208             ctypes.c_float,   # d2 parameter
00209             ctypes.c_float,   # convergence precision tolerance
00210             ctypes.c_int,     # Maximum number of iterations
00211             ctypes.c_float,   # Preference Threshold
00212             ctypes.c_float,   # Veto Threshold
00213             ctypes.c_float,   # Concordance Threshold
00214             ctypes.c_float    # Discordance Threshold
00215         ]
00216 
00217         self.flagr_libflagr_lib.DIBRA.restype = None
00218 
00219     def aggregate(self, input_file="", input_df=None, rels_file="", rels_df=None, out_dir=None):
00220         # This is the directory where the output files are written.  If nothing is provided, then the
    preset temp
00221         # directory of the OS is used.  If an invalid path is provided, the aforementioned temp dir is
    used silently.
00222         if out_dir is not None and os.path.isdir(out_dir):
00223             self.output_diroutput_dir = out_dir
00224 
00225         status = self.check_get_input(input_file, input_df)
00226         if status != 0:
00227             return
00228 
00229         status = self.check_get_rels_input(rels_file, rels_df)
00230         if status != 0:
00231             return
00232 
00233         ran_str = self.get_random_string(16)
00234 
00235         # Call the exposed Agglomerative C function
00236         self.flagr_libflagr_lib.DIBRA(
00237             bytes(self.input_fileinput_file, 'ASCII'),
00238             bytes(self.rels_filerels_file, 'ASCII'),
00239             self.eval_ptseval_pts,
00240             self.aggagg,
00241             bytes(ran_str, 'ASCII'),
00242             bytes(self.output_diroutput_dir, 'ASCII'),
00243             self.weight_normweight_norm,
00244             self.distance_metricdistance_metric,
00245             self.list_pruninglist_pruning,
00246             self.gg,
00247             self.d_1d_1,
00248             self.d_2d_2,
00249             self.tolerancetolerance,
00250             self.mitermiter,
00251             self.preference_tpreference_t,
00252             self.veto_tveto_t,
00253             self.concordance_tconcordance_t,
00254             self.discordance_tdiscordance_t
00255         )
00256 
00257         df_out, df_eval = self.get_output(self.output_diroutput_dir, ran_str)
00258         return df_out, df_eval
```

8.21 PYFLAGR/README.md File Reference

8.22 PYFLAGR/setup.py File Reference

Namespaces

- namespace setup

Variables

- string setup.DESCRIPTION = 'PyFLAGR is a Python package for aggregating ranked preference lists from multiple sources.'
- string setup.LONG_DESCRIPTION
- setup.name
- setup.version
- setup.description
- setup.long_description
- setup.long_description_content_type
- setup.author
- setup.author_email
- setup.maintainer
- setup.maintainer_email
- setup.packages
- setup.url
- setup.install_requires
- setup.license
- setup.keywords
- setup.py_modules
- setup.package_data

8.23 setup.py

Go to the documentation of this file.

```
00001 from distutils.core import setup
00002 from setuptools import find_packages
00003
00004 DESCRIPTION = 'PyFLAGR is a Python package for aggregating ranked preference lists from multiple
      sources.'
00005 LONG_DESCRIPTION = 'The fusion of multiple ranked lists of elements into a single aggregate list is a
      well-studied '\
00006     'research field with numerous applications in Bioinformatics, recommendation systems,
      collaborative filtering, '\
00007     'election systems and metasearch engines.\n\n' \
00008     'FLAGR is a high performance, modular, open source library for rank aggregation problems.  It
      implements baseline '\
00009     'and recent state-of-the-art aggregation algorithms that accept ranked preference lists and
      generate a single '\
00010     'consensus list of elements.  A portion of these methods apply exploratory analysis techniques and
      belong to the '\
00011     'broad family of unsupervised learning techniques.\n\n' \
00012     'PyFLAGR is a Python library built on top of FLAGR library core.  It can be easily installed with
      pip and used in '\
00013     'standard Python programs and Jupyter notebooks.\n\n\n' \
00014     'FLAGR Website:  [https://flagr.site/](https://flagr.site/)\n\n' \
00015     'GitHub repository:
      [https://github.com/lakritidis/FLAGR](https://github.com/lakritidis/FLAGR)\n\n'
00016
00017 setup(
00018     name='pyflagr',
00019     version='1.0.8',
00020     description=DESCRIPTION,
00021     long_description=LONG_DESCRIPTION,
00022     long_description_content_type='text/markdown',
00023     author="Leonidas Akritidis",
00024     author_email="lakritidis@ihu.gr",
00025     maintainer="Leonidas Akritidis",
00026     maintainer_email="lakritidis@ihu.gr",
00027     packages=find_packages(),
00028     url='https://github.com/lakritidis/FLAGR',
00029     install_requires=["pandas", "matplotlib"],
00030     license="Apache",
00031     keywords=[
00032         "rank aggregation", "rank fusion", "data fusion", "unsupervised learning", "information
      retrieval",
00033         "metasearch", "metasearch engines", "borda count", "condorcet", "kendall", "spearman"],
00034     py_modules=["flagr"],
00035     package_data={":  ['flagr.so', 'flagr.dll', 'libgcc_s_seh-1.dll', 'libstdc++-6.dll']}
00036 )
```

Index