R functions for rankings aggregation.

January 16, 2015

Introduction

The following functions were implemented during the computer science master course *"Laboratory of biological data mining"* hold by prof.Balzieri, prof. Moser and prof. Cavecchia on the academic year 2014/2015. The functions can be used for rankings aggregation. In particular they were used to aggregate different ranks that were the output of PC-IM algorithm performed with different input parameters on some gene network. All these functions may be applied on different type of rankings: *top-k-partial*, *total* and *partial rankings*.

Some of the functions are based on the scripts of dott. Andrea Argentini. The other were written by the "rankings" group of students from the course, composed by Luca Erculiani and Caterina Gallo. Caterina wrote the R functions. Luca wrote a python interface that uses those R functions through rpy2. Together tested all the functions for bugs and improvements.

Indice

precision_computator

Description

Giving in input a rank and a table with the class of each item, it returns the precision of the rank. The classes must be divided in four class, with only the first two to be considered as good ones.

Usage

```
precision_computator(dataframe_rank, dataframe_classes, k_firsts
= nrow(dataframe_rank))
```

Arguments

`dataframe_rank`	a dataframe composed by two column, the first one with the items, the second with their ordered rank.
`dataframe_classes`	a dataframe composed by two column, the first one with the items, the second with their respective class.
`k_first`	the first `k` items to be considered. Default value correspond to all the items in the rank in input.

Value

It returns a value that represent the precision of the first `k` rows of the rank.

Note

If `k_first` is different than the number of rows of the dataframe with the rankings, and in the position `k_first` and `k_first+1` we are in the same tile (i.e. subset of items with same rank) every item precision in the intersection between that tile and the `k_first` elements is computed as the precision of the tile.

Author(s)

Caterina Gallo

random_ranker

Description

It takes a set of ranks as an input, and return an aggregation rank composed by random items taken from the input set.

Usage

```
random_ranker(dataframe,col_discarded=0,k_max=(ncol(dataframe) -
col_discarded))
```

Arguments

`dataframe`	a dataframe composed by ranks. Every row is a different rank. The *i-th* element of every rank must be in the same column.
`col_discarded`	how many columns of meta data we have in our dataframe.
`k_max`	the first `k` items to be considered for every rank.

Value

It returns the aggregate rank as a two-columns dataframe. The first column is an ordered set of items, the second column is their respective rank.

Note

`random_ranker` can be used to test if any other ranking aggregator is powerful enough to be at least better than a randomly chosen set of items. Since it is random generated, it is suggested to test its performance as a mean of the performances on more than one run (`10` or `100` for example).

Author(s)

Caterina Gallo

borda_count_ranker

Description

It determines the aggregated rank by giving each item, for each rank, a score based on the position of the item in the rank, and performing a statistic on the set of scores of each item.

Usage

```
borda_count_ranker(dataframe,col_discarded=0,k_max=(ncol(datafra
me) - col_discarded), est=mean)
```

Arguments

`dataframe`	a dataframe composed by ranks. Every row is a different rank. The *i-th* element of every rank must be in the same column.
`col_discarded`	how many columns of meta data we have in our dataframe.
`k_max`	the first `k` items to be considered for every rank.
`est`	the estimator to be used. Possible estimator are functions that can be used on numeric vector. Some examples are: `sum`, `mean`, `min`, `median`.

Value

It returns the aggregate rank as a two-columns dataframe. The first column is an ordered set of items, the second column is their respective rank.

Note

The `borda_count_ranker` assign a score based on the position, higher the position lower the score. So for each statistic, the aggregation is computed taking as higher ranked items the ones with smaller values after the statistic is performed.

Author(s)

Caterina Gallo

mc4_ranker

Description

It computes a transition matrix such that the steady state of the chain assign higher probability to the elements with higher rank. It then ranks the items from the ones with higher probability to the ones with lower probability.

Usage

```
mc4_ranker(dataframe,col_discarded=0,k_max=ncol(dataframe) -
col_discarded, alpha=0.05)
```

Arguments

`dataframe`	a dataframe composed by ranks. Every row is a different rank. The *i-th* element of every rank must be in the same column.
`col_discarded`	how many columns of meta data we have in our dataframe.
`k_max`	the first `k` items to be considered for every rank.
`alpha`	the significance level α to be used (usually α=0.05 and α=0.01).

Value

It returns the aggregate rank as a two-columns dataframe. The first column is an ordered set of items, the second column is their respective rank, based on the steady states probability

Note

The `mc4_ranker` is based on the MC4 heuristic method. For further information see the paper of *C. Dwork, R. Kumar, M. Naor, and D. Sivakumar, "Rank Aggregation Methods for the Web," in Proceedings of the 10-th WWW Conference, pp. 613–622, 2001.*

Author(s)

dott. Andrea Argentini

bba_par_ranker

Description

It aggregates rankings using the Belief Ranking Estimator (BRE). It is based on Belief Function Theory, a general framework for reasoning with uncertainty, with understood connections to other frameworks such as probability, possibility and imprecise probability theories.

Usage

```
bba_par_ranker(dataframe,col_discarded=0,k_max=ncol(dataframe) -
col_discarded, Nite=1, est=mean, MC4=FALSE)
```

Arguments

`dataframe`	a dataframe composed by ranks. Every row is a different rank. The *i-th* element of every rank must be in the same column.
`col_discarded`	how many columns of meta data we have in our dataframe.
`k_max`	the first `k` items to be considered for every rank.
`Nite`	the number of iteration of the algorithm. If `Nite > 1` a penalty weights are applied. Possibly the worst ranker is substituted for `Nite-1` times.
`est`	estimator used to estimate the true rank. Here function as `mean`, `min`, `median` or function applicable on numeric vectors can be used.
`MC4`	boolean variable set as `FALSE`. When set to `TRUE`, the parameter `est` is ignored and the estimation of the true rank is done by the heuristic MC4.

Value

It returns the aggregate rank as a two-columns dataframe. The first column is an ordered set of items, the second column is their respective rank.

Note

For further information see *A.Argentini, E. Blanzieri, "Ranking Estimation with Belief Functions"*

Author(s)

dott. Andrea Argentini

---

other functions

Other functions in the script are used in order to compute the belief ranking estimator. Particularly interesting may be `bba.b` that compute the weights used on the `bba_par_ranker`. Some functions considered here were $1-(x/k_{max})^2$ or $(x/k_{max}-1)^2$ but without great improvement. Also weights using the relative frequencies of each gene in each rank were considered.

`compute_MC4` is really similar to `mc4_ranker`. It is used when we need in the `bba_par_ranker` the heuristic MC4 as an estimator of the true rank.