Package ‘SuperRanker’

January 29, 2021

Type Package
Title Sequential Rank Agreement
Version 1.2.0
Date 2021-01-29
Author Claus Thorn Ekstrøm and Thomas Alexander Gerds
Maintainer Claus Thorn Ekstrøm <ekstrom@sund.ku.dk>
Description Tools for analysing the agreement of two or more rankings of the same items. Examples are importance rankings of predictor variables and risk predictions of subjects. Benchmarks for agreement are computed based on random permutation and bootstrap.
License GPL (>= 2)
Imports stats, graphics, Rcpp (>= 0.11.5), prodlim (>= 1.5.7)
LinkingTo Rcpp
Encoding UTF-8
ByteCompile true
RoxygenNote 7.1.1
Suggests testthat (>= 3.0.0)
Config/testthat/edition 3
NeedsCompilation yes
Repository CRAN
Date/Publication 2021-01-29 22:10:02 UTC

R topics documented:

average_overlap .................................................. 2
overlap .................................................................. 2
plot.sra ............................................................... 3
random_list_sra .................................................... 4
smooth_sra .......................................................... 5
sra ..................................................................... 6
sracpp .................................................................. 8
average_overlap

Description
Compute the average overlap

Usage
average_overlap(obj)

Arguments
obj Either a vector or matrix

Value
A vector of the average overlap

Examples
# setting with 3 lists
mlist <- matrix(cbind(1:8,c(1,2,3,5,6,7,4,8),c(1,5,3,4,2,8,7,6)),ncol=3)
average_overlap(mlist)

overlap

Description
Computes the overlap (number of items present in all k lists divided by the current rank) for each rank in the k lists

Usage
overlap(rankMat)

Arguments
rankMat A matrix with k columns corresponding to the k ranked lists. Elements of each column are integers between 1 and the length of the lists
plot.sra

**Value**

A vector of the same length as the rows in rankMat containing the overlap between the lists for each rank

**Author(s)**

Claus Ekstrøm <ekstrom@sund.ku.dk>

---

**plot.sra**

*Plot sequential rank agreement*

**Description**

Plot the agreement between lists as a function of the list depth

**Usage**

```r
## S3 method for class 'sra'
plot(
x,  
xlim,  
ylim,  
xlab = "List depth",  
ylab = "Sequential rank agreement",  
add = FALSE,  
...)
```

**Arguments**

- `x` Agreement object
- `xlim` x-axis limits
- `ylim` y-axis limits
- `xlab` x-axis lab
- `ylab` y-axis lab
- `add` Logical. If TRUE add graph to existing plot.
- `...` Processed by function prodlim::SmartControl.

**Value**

Graph

**Author(s)**

Thomas A. Gerds <tag@biostat.ku.dk>
random_list_sra

Simulate sequential rank agreement for randomized unrelated lists

Description

Simulate sequential rank agreement from completely uninformative lists (i.e., raw permutations of items) and compute the corresponding sequential rank agreement curves. The following attributes are copied from the input object: number of lists, number of items and amount of censoring.

Usage

random_list_sra(object, B = 1, n = 1, na.strings = NULL, nitems = NULL, type = c("sd", "mad"), epsilon = 0)

Arguments

object A matrix of numbers or list of vectors representing ranked lists.
B An integer giving the number of randomizations to sample over in the case of censored observations
n Integer: the number of permutation runs. For each permutation run we permute each of the lists in object and compute corresponding the sequential rank agreement curves
### Description

Smooth quantiles of a matrix of sequential ranked agreements.

### Usage

```
smooth_sra(object, confidence = 0.95)
```

### Arguments

- `object` A matrix
- `confidence` the limits to compute

### Value

A list containing two vectors for the smoothed lower and upper limits
Examples

# setting with 3 lists
mlist <- matrix(cbind(1:8,c(1,2,3,5,6,7,4,8),c(1,5,3,4,2,8,7,6)),ncol=3)
# compute rank agreement of 5 random permutations
null=random_list_sra(mlist,n=15)
# now extract point-wise quantiles according to confidence level
smooth_sra(null)

sra

Compute the sequential rank agreement

Description

Compute the sequential rank agreement

Usage

sra(object, B, na.strings, nitems, type, epsilon = 0, ...)

## Default S3 method:
sra(object, B, na.strings, nitems, type, epsilon = 0, ...)

## S3 method for class 'matrix'
sra(
  object,
  B = 1,
  na.strings = NULL,
  nitems = nrow(object),
  type = c("sd", "mad"),
  epsilon = 0,
  ...
)

## S3 method for class 'list'
sra(
  object,
  B = 1,
  na.strings = NULL,
  nitems = max(sapply(object, length)),
  type = c("sd", "mad"),
  epsilon = 0,
  ...
)
Arguments

object Either matrix where each column is a ranked list of items or a list of ranked lists of items. Elements are integers between 1 and the length of the lists. The lists should have the same length but censoring can be used by setting the list to zero from a point onwards. See details for more information.

B An integer giving the number of randomization to sample over in the case of censored observations

na.strings A vector of strings/values that represent missing values in addition to NA. Defaults to NULL which means only NA are censored values.

nitems The total number of items in the original lists if we only have partial lists available.

type The type of measure to use. Either sd (standard deviation - the default) or mad (median absolute deviance around the median)

epsilon A non-negative numeric vector that contains the minimum limit in proportion of lists that must show the item. Defaults to 0. If a single number is provided then the value will be recycles to the number of items.

... Arguments passed to methods.

Value

A vector of the sequential rank agreement

Author(s)

Claus Ekstrøm <ekstrom@sund.ku.dk> and Thomas A Gerds <tag@biostat.ku.dk>

Examples

mlist <- matrix(cbind(1:8,c(1,2,3,5,6,7,4,8),c(1,5,3,4,2,8,7,6)),ncol=3)
sra(mlist)

mlist <- matrix(cbind(1:8,c(1,2,3,5,6,7,4,8),c(1,5,3,4,2,8,7,6)),ncol=3)
sra(mlist, nitems=20, B=10)

alist <- list(a=1:8,b=sample(1:8),c=sample(1:8))
sra(alist)

blist <- list(x1=letters,x2=sample(letters),x3=sample(letters))
sra(blist)

## censored lists are either too short
clist <- list(x1=c("a","b","c","d","e","f","g","h"),
   x2=c("h","c","f","g","b"),
   x3=c("d","e","a"))
set.seed(17)
sra(clist,na.strings="z",B=10)
## or use a special code for missing elements

```r
Clist <- list(x1=c("a","b","c","d","e","f","g","h"),
               x2=c("h","c","f","g","b","z","z","z"),
               x3=c("d","e","a","z","z","z","z","z"))
```

```r
set.seed(17)
sra(Clist,na.strings="z",B=10)
```

---

### sracpp

**Compute the sequential rank agreement between k ranked lists**

#### Description

Computes the sequential rank agreement (number of items present in all k lists divided by the current rank) for each rank in the k lists.

#### Usage

```r
sracpp(rankMat, maxlength, B, cens, type = 0L, epsilon = as.numeric(c(0)))
```

#### Arguments

- `rankMat`: A matrix with k columns corresponding to the k ranked lists. Elements of each column are integers between 1 and the length of the lists.
- `maxlength`: The maximum depth that are needed.
- `B`: The number of resamples to use in the presence of censored lists.
- `cens`: A vector of integer values that.
- `type`: The type of distance measure to use: 0 (the default) is the variance while 1 is MAD (median absolute deviation).
- `epsilon`: A non-negative numeric vector that contains the minimum limit in proportion of lists that must show the item. Defaults to 0. If a single number is provided then the value will be recycles to the number of items.

#### Value

A vector of the same length as the rows in rankMat containing the squared (!) sequential rank agreement between the lists for each depth. If the MAD type was chosen then the sequential MAD values are returned.

#### Author(s)

Claus Ekstrøm <ekstrom@sund.ku.dk>
sracppfull

Compute the sequential rank agreement between k ranked lists

Description
Computes the sequential rank agreement (number of items present in all k lists divided by the current rank) for each rank in the k lists.

Usage
sracppfull(rankMat, type = 0L, epsilon = as.numeric(c(0)))

Arguments
- rankMat: A matrix with k columns corresponding to the k ranked lists. Elements of each column are integers between 1 and the length of the lists.
- type: The type of distance measure to use: 0 (the default) is the variance while 1 is MAD (mean absolute deviation).
- epsilon: A non-negative numeric vector that contains the minimum limit in proportion of lists that must show the item. Defaults to 0. If a single number is provided then the value will be recycled to the number of items.

Value
A vector of the same length as the rows in rankMat containing the sequential rank agreement between the lists for each depth (squared for type=0).

Author(s)
Claus Ekstrøm <ekstrom@sund.ku.dk>

SuperRanker

Functions related to comparison of ranked lists

Description
SuperRanker allows you to estimate the agreement between two or more rankings of the same items.
test_sra

*Compute a Kolmogorov-Smirnoff-like test for Smooth quantiles of a matrix of sequential rank agreements*

**Description**

Compute a Kolmogorov-Smirnoff-like test for Smooth quantiles of a matrix of sequential rank agreements

**Usage**

```r
test_sra(object, nullobject, weights = 1)
```

**Arguments**

- `object` An object created with `sra`.
- `nullobject` An object created with `random_list_sra`.
- `weights` Either a single value or a vector of the same length as the number of item with the weight that should be given to specific depths.

**Value**

A single value corresponding to the p-value

**Author(s)**

Claus Ekstrøm <ekstrom@sund.ku.dk>

**Examples**

```r
# setting with 3 lists
mlist <- matrix(cbind(1:8,c(1,2,3,5,6,7,4,8),c(1,5,3,4,2,8,7,6)),ncol=3)
# compute sequential rank agreements
x <- sra(mlist)
# compute rank agreement of 5 random permutations
null <- random_list_sra(mlist,n=15)
# now extract point-wise quantiles according to confidence level
test_sra(x,null)
# compare to when we use the result of the first permutation run
test_sra(null[,1],null[,1])
```
Index

average_overlap, 2
overlap, 2
plot.sra, 3
random_list_sra, 4
smooth_sra, 5
sra, 6
sracpp, 8
sracppfull, 9
SuperRanker, 9
test_sra, 10