Package 'rapidsplit'

June 14, 2024

Type Package

Title Fast split-half reliability algorithm

Version 0.2

Date 2021-11-24

Author Sercan Kahveci

Maintainer Sercan Kahveci <sercan.kahveci@plus.ac.at>

Description A fast and flexible split-half reliability algorithm.

License GPL (>= 2)

Depends R(>= 3.5)

Imports Rcpp (>= 1.0.5)

LinkingTo Rcpp

RoxygenNote 7.3.1

Encoding UTF-8

Contents

bootstrapWeights *Bootstrap Weights*

Description

Create a matrix of bootstrap samples expressed as frequency weights

Usage

```
bootstrapWeights(size, times)
```

Arguments

size	Number of values to bootstrap
times	Number of bootstraps

Value

A matrix with bootstrap samples expressed as frequency weights. Each column represents a single bootstrap iteration and each row represents a case.

Examples

```
# Rapidly compute a bootstrapped median to obtain its standard error
myweights<-bootstrapWeights(size=50, times=100)
meds<-mediansByWeight(x=rnorm(50),weights=myweights)
# SE
sd(meds)
```

colAggregators *Fast matrix column aggregators*

Description

Fast matrix column aggregators

Usage

```
colMedians(x)

colSds(x)

colMediansMasked(x, mask)

colMeansMasked(x, mask)

colSdsMasked(x, mask)
```

Arguments

x	A numeric matrix to compute column aggregates of.
mask	A logical matrix determining which data points to include in the column-wise aggregations.

See Also

colMeans, mediansByMask, maskAggregators

Examples

```
x <- cbind(x1 = 3, x2 = c(4:1, 2:5))
colMedians(x)

colSds(x)

mask<-cbind(rep(c(TRUE,FALSE),4),
            rep(c(TRUE,FALSE),each=4))
colMediansMasked(x,mask)

colMeansMasked(x,mask)

colSdsMasked(x,mask)
```

---

corByColumns *Correlate two matrices by column*

Description

Correlate each column of 1 matrix with the same column in another matrix

Usage

```
corByColumns(x, y)

corByColumns_mask(x, y, mask)
```

Arguments

x, y	Matrices whose values to correlate by column.
mask	Logical matrix marking which data points to include.

Details

The primary use for these functions is to rapidly compute the correlations between two sets of split-half scores stored in matrix columns.

Value

A numeric vector of correlations per column.

Examples

```
m1<-matrix((1:9)+rnorm(9),ncol=3)
m2<-matrix((9:1)+rnorm(9),ncol=3)
corByColumns(m1,m2)

mask<-1-diag(3)
corByColumns_mask(m1,m2,mask)
```

`cormean` *Compute a minimally biased average of correlation values*

Description

This function computes a minimally biased average of correlation values. This is needed because simple averaging of correlations is negatively biased, and the often used z-transformation method of averaging correlations is positively biased. The algorithm was developed by Olkin & Pratt (1958).

Usage

```
cormean(
  r,
  n,
  weights = c("none", "n", "df"),
  type = c("OP5", "OP2", "OPK"),
  na.rm = F
)
```

Arguments

`r`	A vector containing correlation values/
`n`	A single value or vector containing sample sizes/
`weights`	Character. How should the correlations be weighted? `none` leads to no weighting, `n` weights by sample size, `df` weights by sample size minus one.
`type`	Character. Determines which averaging algorithm to use, with "OP5" usually being the most accurate.
`na.rm`	Logical. Should missing values be removed?

Value

An average correlation.

References

Olkin, I., & Pratt, J. (1958). Unbiased estimation of certain correlation coefficients. The Annals of Mathematical Statistics, 29. https://doi.org/10.1214/aoms/1177706717

Shieh, G. (2010). Estimation of the simple correlation coefficient. Behavior Research Methods, 42(4), 906-917. https://doi.org/10.3758/BRM.42.4.906

Examples

```
cormean(c(0,.3,.5),c(30,30,60))
```

`correlation-tools` *Correlation tools*

Description

Helper functions to compute important statistics from correlation coefficients.

Usage

```
r2z(r)

z2r(z)

r2t(r, n)

t2r(t, n)

r2p(r, n)

rconfint(r, n, alpha = 0.05)

compcorr(r1, r2, n1, n2)

## S3 method for class 'compcorr'
print(x, ...)
```

Arguments

`r`, `r1`, `r2`	Correlation values.
`z`	Z-scores.
`n`, `n1`, `n2`	Sample sizes.
`t`	t-scores.
`alpha`	The significance level to use.
`x`	A `compcorr` object to print.
`...`	Ignored.

Functions

- `r2z()`: Converts correlation coefficients to z-scores.
- `z2r()`: Converts z-scores to correlation coefficients.
- `r2t()`: Converts correlation coefficients to t-scores.
- `t2r()`: Converts t-scores to correlation coefficients.
- `r2p()`: Computes the two-sided p-value for a given correlation.
- `rconfint()`: Computes confidence intervals for a given correlation coefficient.

- `compcorr()`: Computes the significance of the difference between two correlation coefficients.
- `print(compcorr)`: Computes the significance of the difference between two correlation coefficients.

See Also

cormean

Examples

```
z <- r2z(.5)
r <- z2r(z)
t<-r2t(r,30)
r<-t2r(t,30)
r2p(r,30)
print(rconfint(r,30))
print(compcorr(.5,.7,20,20))
```

`excludeOutliersByMask` *Exclude SD-based outliers*

Description

Different masks (columns of a logical matrix) are applied to the same input vector, and outliers in each resulting subvector are marked with `FALSE` in the mask.

Usage

```
excludeOutliersByMask(x, mask, sdlim = 3)
```

Arguments

`x`	Vector to exclude outliers from.
`mask`	A logical matrix determining which data points to include and which not to.
`sdlim`	Standard deviation limit to apply; values beyond are classified as outliers and masked.

Value

An updated mask.

Examples

```
x<-rnorm(50)
x[1]<-100
x[2]<-50
mask<-matrix(TRUE,ncol=3,nrow=50)
mask[1,2]<-FALSE
mask[2,3]<-FALSE
excludeOutliersByMask(x,mask)
```

foodAAT — *Approach-Avoidance Task examining approach bias to different foods*

Description

This data originates from an approach-avoidance task examining approach bias towards food. Participants responded to the stimulus category (food or object) by pulling or pushing a joystick. Instructions were flipped from one block to the next.

Usage

```
data(foodAAT)
```

Format

An object of class `"data.frame"`

Details

* subjectid: Participant ID * stimid: Stimulus ID * is_pull: Whether the trial required an approach response (1) or an avoid response (0) * is_target: Whether the trial featured a food stimulus (1) or an object stimulus (0) * error: Whether the response was incorrect (1) or correct (0) * RT: The response initiation time * FullRT: The time from stimulus onset to response completion * trialnum: The trial number * blocknum: The block number * palatability: The participant's palatability rating for the stimulus (foods only) * valence: The participant's valence rating for the stimulus * FCQS_2_craving: The participant's FCQS state food craving score at time of testing * FCQS_2_hunger: The participant's FCQS state hunger score at time of testing

Source

Original study

References

Lender, A., Meule, A., Rinck, M., Brockmeyer, T., & Blechert, J. (2018). Measurement of food-related approach–avoidance biases: Larger biases when food stimuli are task relevant. Appetite, 125, 42–47. 10.1016/j.appet.2018.01.032

maskAggregators — *Multi-mask/weight based aggregators*

Description

Methods to aggregate the same vector with different masks or frequency weights. Useful for fast bootstrapping or split-half scoring. A single aggregate value of `x` is computed for each column of the mask or weight matrix.

Usage

```
mediansByMask(x, mask)

meansByMask(x, mask)

sdsByMask(x, mask)

mediansByWeight(x, weights)

meansByWeight(x, weights)

sdsByWeight(x, weights)
```

Arguments

x	A vector to aggregate over with different masks or weights.
mask	Logical matrix where each column represents a separate vector of masks to aggregate x with. Only values marked TRUE are included in the aggregation.
weights	Integer matrix where each column represents frequency weights to weight the aggregation by.

See Also

colMedians, colAggregators

Examples

```
# Demonstration of mediansByMask()
x<-1:6
mask<-rbind(c(TRUE,FALSE,FALSE),
            c(TRUE,FALSE,FALSE),
            c(FALSE,TRUE,FALSE),
            c(FALSE,TRUE,FALSE),
            c(FALSE,FALSE,TRUE),
            c(FALSE,FALSE,TRUE))
mediansByMask(x,mask)

# Compute split-halves for a single
# participant, stratified by stimulus
data(foodAAT)
currdata<-foodAAT[foodAAT$subjectid==3,]
currdata$stratfactor<-
  interaction(currdata$is_pull,
              currdata$is_target,
              currdata$stimid)
currdata<-currdata[order(currdata$stratfactor),]
groupsizes<-
  rle(as.character(currdata$stratfactor))$lengths
mysplits<-
  stratifiedItersplits(splits=1000,
                       groupsizes=groupsizes)

# Median for half 1
mediansByMask(currdata$RT,mysplits==1)
```

```
#How to use meansByMask()
meansByMask(x,mask)
sd(meansByMask(currdata$RT,mysplits==1))

# How to use sdsByMask() to compute
# mask-based D-scores
meansByMask(currdata$RT,mysplits==1) /
  sdsByMask(currdata$RT,mysplits==1)

# Compute the bootstrapped
# standard error of a median
weights<-
  bootstrapWeights(size=nrow(currdata),
                   times=1000)
bootmeds<-mediansByWeight(currdata$RT,weights)
sd(bootmeds) # bootstrapped standard error

# Compute the bootstrapped
# standard error of a mean
bootmeans<-meansByWeight(currdata$RT,weights)
sd(bootmeans) # bootstrapped standard error
# exact standard error for comparison
sd(currdata$RT)/sqrt(length(currdata$RT))

# Use sdsByWeight to compute bootstrapped D-scores
bootsds<-sdsByWeight(currdata$RT,weights)
# bootstrapped standard error of D-score
sd(bootmeans/bootsds)
```

`OutlierMaskers` *Exclude SD-based outliers in each matrix column*

Description

Generate or update a mask matrix based on outlyingness of values in each column.

Usage

```
maskOutliers(x, sdlim = 3)

maskOutliersMasked(x, mask, sdlim = 3)
```

Arguments

`x`	Matrix in which to mark SD-based outliers by column.
`sdlim`	Standard deviation limit to apply; values beyond are classified as outliers and masked.
`mask`	A logical matrix determining which data points to include and which not to.

Value

A logical matrix with outliers (and previously masked values) marked as `FALSE`.

Examples

```
# Generate data with outliers
testmat<-matrix(rnorm(100),ncol=2)
testmat[1,]<-100
testmat[2,]<-50

# Detect outliers
maskOutliers(testmat)

# Generate a mask
testmask<-matrix(T,ncol=2,nrow=50)
testmask[1,1]<-F

# Detect outliers with pre-existing mask
maskOutliersMasked(x=testmat,
                   mask=testmask, sdlim = 3)
```

`rapidsplit` *rapidsplit*

Description

A very fast algorithm for computing stratified permutated split-half reliability.

Usage

```
rapidsplit(
  data,
  subjvar,
  diffvars = NULL,
  stratvars = NULL,
  aggvar,
  splits,
  aggfunc = c("means", "medians"),
  standardize = FALSE,
  include.scores = TRUE
)

## S3 method for class 'rapidsplit'
print(x, ...)

## S3 method for class 'rapidsplit'
plot(x, type = c("average", "minimum", "maximum", "random", "all"), ...)
```

Arguments

`data`	Dataset, a `data.frame`.
`subjvar`	Subject ID variable name, a `character`.
`diffvars`	Names of variables that determine which conditions need to be subtracted from each other, `character`.

stratvars	Additional variables that the splits should be stratified by; a character.
aggvar	Name of variable whose values to aggregate, a character. Examples include reaction times and error rates.
splits	Number of split-halves to average, an integer.
aggfunc	The function by which to aggregate the variable defined in aggvar; can be "means" or "medians".
standardize	Whether to divide by scores by the subject's SD; a logical.
include.scores	Include all individual split-half scores?
x	rapidsplit object to print or plot.
...	Ignored.
type	Character argument indicating what should be plotted. By default, this plots the random split whose correlation is closest to the average. However, this can also plot the random split with the "minimum" or "maximum" split-half correlation, or any "random" split. "all" splits can also be plotted together in one figure.

Value

A list containing

* r: the averaged reliability.

* allcors: a vector with the reliability of each iteration.

* nobs: the number of participants.

* scores: the individual participants scores in each split-half, contained in a list with two matrices (Only included if requested with include.scores).

Examples

```
data(foodAAT)
# Reliability of the double difference score:
# [RT(push food)-RT(pull food)] - [RT(push object)-RT(pull object)]

frel<-rapidsplit(data=foodAAT,
                 subjvar="subjectid",
                 diffvars=c("is_pull","is_target"),
                 stratvars="stimid",
                 aggvar="RT",
                 splits=1000,
                 aggfunc="mean")

print(frel)

plot(frel,type="all")

# Unstratified reliability of the median RT
rapidsplit(data=foodAAT,
           subjvar="subjectid",
           aggvar="RT",
           splits=1000,
           aggfunc="median")
```

SpearmanBrown *Spearman-Brown correction Perform a Spearman-Brown correction on the provided correlation score.*

Description

Spearman-Brown correction Perform a Spearman-Brown correction on the provided correlation score.

Usage

```
SpearmanBrown(r, ntests = 2)
```

Arguments

r	To-be-corrected correlation coefficient.
ntests	An integer indicating how many times larger the full test is, for which the corrected correlation coefficient is being computed.

Details

When ntests=2, the formula will compute what the correlation coefficient would be if the test were twice as long.

Value

Spearman-Brown corrected correlation coefficients.

Examples

```
SpearmanBrown(.5)
```

stratifiedItersplits *stratifiedItersplits*

Description

Generate stratified splits for a single participant

Usage

```
stratifiedItersplits(splits, groupsizes)
```

Arguments

splits	Number of iterations.
groupsizes	An integer vector of how many RTs per group need to be stratified.

Details

This equally splits what can be equally split within groups. Then it randomly splits all the leftovers to ensure near-equal split sizes. This function is moreso used internally, but you can use it if you know what you are doing.

Value

A matrix with zeroes and ones. Each column is a random split.

Examples

```
# We will create splits stratified by stimulus for a single participant
data(foodAAT)
currdata<-foodAAT[foodAAT$subjectid==3,]
currdata$stratfactor<-interaction(currdata$is_pull,currdata$is_target,currdata$stimid)
currdata<-currdata[order(currdata$stratfactor),]
groupsizes<-rle(as.character(currdata$stratfactor))$lengths

mysplits<-stratifiedItersplits(splits=1000,groupsizes=groupsizes)

# Now the data can be split with the values from any column.
half1<-currdata[mysplits[,1]==1,]
half2<-currdata[mysplits[,1]==0,]

# Or the split objects can be used as masks for the aggregation functions in this package
meansByMask(x=currdata$RT,mask=mysplits==1)
```

Index