Package 'IATscores'

July 24, 2019

Title Implicit Association Test Scores Using Robust Statistics

Description Compute several variations of the Implicit Association Test (IAT) scores, including the D scores (Greenwald, Nosek, Banaji, 2003, <doi:10.1037/0022-3514.85.2.197>) and the new scores that were developed using robust statistics (Richetin, Costantini, Perugini, and Schonbrodt, 2015, <doi:10.1371/journal.pone.0129601>).

Author Giulio Costantini

Maintainer Giulio Costantini <costantinigiulio@gmail.com>

Date 2019-07-24

Version 0.2.3

Type Package

Depends R (>= 3.4.0)

Imports stringr (>= 1.2.0), dplyr (>= 0.7.2), reshape2 (>= 1.4.2), qgraph (>= 1.4.4), methods (>= 3.4.1)

Suggests nparcomp (>= 2.6)

License GPL-2

NeedsCompilation no

Repository CRAN

Date/Publication 2019-07-24 21:30:14 UTC

R topics documented:

IATscores-package 1
alg2param 2
IATdescriptives 3
Pretreatment 5
RobustScores 8
SplitHalf 14
TestRetest 17
Tgraph 20

IATscores-package *Compute Robust IAT scores*

Description

The function `RobustScores` computes variants of the robust IAT scores according to four main parameters.

Details

Package:	IATscores
Type:	Package
Version:	0.2.3
Date:	2019-07-05
License:	GPL-2

alg2param *Convert the algorithm names to the generating parameters*

Description

Starting from the algorithm names, gives the parameters that generated each algorithm as output.

Usage

```
alg2param(x)
```

Arguments

`x`	The name of an algorithm (string) or the name of many algorithms (vector of strings).

Details

The algorithm names in this package follow a precise convention and are in the form `"pxxxx"`, (where each `x` stands for a numbers). The first number corresponds to the value of the parameter `P1` in RobustScores, the second number corresponds to the value of `P2` and so on. This function allows to know the values of the parameters that generated an algorithm from the algorithm's name. Also a vector of algorithm's names can be given as input.

Value

A dataframe with four columns.

`algorithm`	(string). The algorithm's name given as input
`P1`	(string). Parameter P1, see `RobustScores`
`P2`	(string). Parameter P2, see `RobustScores`
`P3`	(string). Parameter P3, see `RobustScores`
`P4`	(string). Parameter P4, see `RobustScores`

Author(s)

Giulio Costantini

Examples

```
alg2param("p1231")
```

`IATdescriptives` *Summary statistics of reaction time and error*

Description

Provides several summary statistics for reaction times and errors, by subject and by block. If by block, only two critical blocks, pair1 and pair2, are considered. See function `Pretreatment`).

Usage

```
IATdescriptives(IATdata, byblock = FALSE)
```

Arguments

`IATdata`	a dataframe with the following columns: • `subject`: (factor or coercible to factor). Univocally identifies a participant. • `correct`: (logical). has value `TRUE` or `1` if the trial was answered correctly, `FALSE` or `0` otherwise. • `latency`: (numeric). Response latency, in ms. • `blockcode`: (factor or string). Can assume only two values, `"pair1"` and `"pair2"`. `"pair1"` is for one critical block and `"pair2"` is the other critical block. • `praccrit`. (factor, optional). Can assume only two values, `"prac"` is for practice combined categorization block and `"crit"` is for critical combined categorization block. In a IAT with 60 trials for each double categorization block, the first 20 are sometimes administered as practice block, the other 40 as critical.
`byblock`	If `TRUE`, summary statistics are returned separately for the two critical blocks, pair1 and pair2

Details

These summary statistics are used sometimes to define exclusion criteria. For example, Greenwald, Nosek, & Banaji's (2003) improved algorithm suggests to eliminate subjects for whom more than 10 percent trials have latency less than 300ms.

Value

`Ntrials` number of trials

`Nmissing_latency`
number of trials in which latency information is missing

`Nmissing_accuracy`
number of trials in which accuracy information is missing

`Prop_error` proportion of error trials

`M_latency` mean latency

`SD_latency` SD of latency

`min_latency` minimum value of latency

`max_latency` maximum value of latency

`Prop_latency300`
proportion of latencies faster than 300 ms

`Prop_latency400`
proportion of latencies faster than 400 ms

`Prop_latency10s`
proportion of latencies slower than 10 seconds

Author(s)

Giulio Costantini

References

Greenwald, A. G., Nosek, B. A., & Banaji, M. R. (2003). Understanding and using the Implicit Association Test: I. An improved scoring algorithm. Journal of Personality and Social Psychology, 85(2), 197-216. doi:10.1037/0022-3514.85.2.197

See Also

`Pretreatment`

Examples

```
#### generate random IAT data ####
set.seed(1234)
rawIATdata <- data.frame(
  # ID of each participant (N = 10)
  ID = rep(1:10, each = 180),
```

```
  # seven-block structure, as in Greenwald, Nosek & Banaji (2003)
  # block 1 = target discrimination (e.g., Bush vs. Gore items)
  # block 2 = attribute discrimination (e.g., Pleasant words vs. unpleasant)
  # block 3 = combined practice (e.g., Bush + pleasant vs. Gore + unpleasant)
  # block 4 = combined critical  (e.g., Bush + pleasant vs. Gore + unpleasant)
  # block 5 = reversed target discrimination (e.g., Gore vs. Bush)
  # block 6 = reversed combined practice (e.g., Gore + pleasant vs. Bush + unpleasant)
  # block 7 = reversed combined critical (e.g., Gore + pleasant vs. Bush + unpleasant)
  block = rep(c(rep(1:3, each = 20),
                rep(4, 40),
                rep(5:6, each = 20),
                rep(7, 40)), 10),
  # expected proportion of errors = 10 percent
  correct = sample(c(0, 1), size = 1800, replace = TRUE, prob = c(.2, .8)),
  # reaction times are generated from a mix of two chi2 distributions,
  # one centered on 550ms and one on 100ms to simulate fast latencies
  latency = round(sample(c(rchisq(1500, df = 1, ncp = 550),
                           rchisq(300, df = 1, ncp = 100)), 1800)))

# add some IAT effect by making trials longer in block 6 and 7
rawIATdata[rawIATdata$block >= 6, "latency"] <-
  rawIATdata[rawIATdata$block >= 6, "latency"] + 100

# add some more effect for subjects 1 to 5
rawIATdata[rawIATdata$block >= 6 &
             rawIATdata$ID <= 5, "latency"] <-
  rawIATdata[rawIATdata$block >= 6 &
             rawIATdata$ID <= 5, "latency"] + 100

#### pretreat IAT data using function Pretreatment ####
IATdata <- Pretreatment(rawIATdata,
                           label_subject = "ID",
                         label_latency = "latency",
                         label_accuracy = "correct",
                         label_block = "block",
                         block_pair1 = c(3, 4),
                         block_pair2 = c(6, 7),
                         label_praccrit = "block",
                         block_prac = c(3, 6),
                         block_crit = c(4, 7))
IATdescriptives(IATdata)
```

`Pretreatment` *Pretreat the IAT data in input.*

Description

Convert the initial dataframe of the IAT in a simpler dataframe, which is the input of subsequent functions in this package.

Usage

```
Pretreatment(IATdata,
  label_subject = "subject",
  label_latency = "latency",
  label_accuracy = "correct",
  label_block = "blockcode",
  block_pair1 = c("pair1_left", "pair1_right"),
  block_pair2 = c("pair2_left", "pair2_right"),
  label_trial = NA,
  trial_left = NA,
  trial_right = NA,
  label_praccrit=NA,
  block_prac=NA,
  block_crit=NA,
  label_stimulus=NA)
```

Arguments

`IATdata`	The input dataframe. I consider the the output of the IAT implemented in Inquisit (a row by trial). Only 7 columns are important for computation. - a column with subject numbers - a column with latencies - a column with accuracy (1 = correct, 0 = incorrect) - a column including the block codes, i.e. one or more strings that describe the kind of block (e.g., "compatible" vs. "incompatible") - a column including the trial codes, i.e. one or more strings that describe the kind of trial (e.g., "response_left" vs. "response_right") - a column including information about which are the practice and which the critical combined categorization blocks. - a column with the original stimuli (optional)
`label_subject`	String. Name of the column in `IATdata` with the subject numbers
`label_latency`	String. Name of the column in `IATdata` with the latencies
`label_accuracy`	String. Name of the column in `IATdata` with the accuracy
`label_block`	String. Name of the column in `IATdata` with the block names
`block_pair1`	Vector of strings. Elements of the column indicated in `label_block` that correspond the one of the critical blocks of the IAT
`block_pair2`	Vector of strings. Elements of the column indicated in `label_block` that correspond the the other critical block of the IAT (with respect to the one indicated by `block_pair1`)
`label_trial`	String (optional). Name of the column in `IATdata` with the trial names
`trial_left`	Vector of strings(optional). Elements of the column indicated in `label_trial` that correspond to trials that required to to press the left button to give the correct response.

`trial_right` Vector of strings(optional). Elements of the column indicated in `label_trial` that correspond to trials that required to to press the right button to give the correct response.

`label_praccrit` String (optional). The column in which the information about practice and critical trials is stored.

`block_prac` Vector of strings (optional). The elements of the column indicated in `label_praccrit` that correspond to the practice combined blocks

`block_crit` Vector of strings (optional). The elements of the column indicated in `label_praccrit` that correspond to the critical combined blocks

`label_stimulus` (optional) The variable name in `IATdata` that keeps information about the stimulus presented in each trial

Value

a dataframe with the following columns:

`subject` Univocally identifies a participant.

`correct` (logical). has value TRUE or 1 if the trial was answered correctly, FALSE or 0 otherwise.

`latency` (numeric). Response latency.

`blockcode` (factor). Can assume only two values, `"pair1"` and `"pair2"`. `"pair1"` is for one critical block and `"pair2"` is the other critical block.

`praccrit` (factor, optional). Can assume only two values, `"prac"` is for practice combined categorization block and `"crit"` is for critical combined categorization block. In a IAT with 60 trials for each double categorization block, the first 20 are sometimes administered as practice block, the other 40 as critical.

`trialcode` (factor, optional). Code for the trial, has value `"left"` if the correct response required to press the left button, `"right"` if it required to press the right button.

`stimulus` (character, optional). The stimulus item.

Author(s)

Giulio Costantini

Examples

```
#### generate random IAT data ####
set.seed(1234)
rawIATdata <- data.frame(
  # ID of each participant (N = 10)
  ID = rep(1:10, each = 180),
  # seven-block structure, as in Greenwald, Nosek & Banaji (2003)
  # block 1 = target discrimination (e.g., Bush vs. Gore items)
  # block 2 = attribute discrimination (e.g., Pleasant words vs. unpleasant)
  # block 3 = combined practice (e.g., Bush + pleasant vs. Gore + unpleasant)
```

```
  # block 4 = combined critical  (e.g., Bush + pleasant vs. Gore + unpleasant)
  # block 5 = reversed target discrimination (e.g., Gore vs. Bush)
  # block 6 = reversed combined practice (e.g., Gore + pleasant vs. Bush + unpleasant)
  # block 7 = reversed combined critical (e.g., Gore + pleasant vs. Bush + unpleasant)
  block = rep(c(rep(1:3, each = 20),
                rep(4, 40),
                rep(5:6, each = 20),
                rep(7, 40)), 10),
# expected proportion of errors = 10 percent
  correct = sample(c(0, 1), size = 1800, replace = TRUE, prob = c(.2, .8)),
  # reaction times are generated from a mix of two chi2 distributions,
  # one centered on 550ms and one on 100ms to simulate fast latencies
  latency = round(sample(c(rchisq(1500, df = 1, ncp = 550),
                           rchisq(300, df = 1, ncp = 100)), 1800)))

# add some IAT effect by making trials longer in block 6 and 7
rawIATdata[rawIATdata$block >= 6, "latency"] <-
  rawIATdata[rawIATdata$block >= 6, "latency"] + 100

# add some more effect for subjects 1 to 5
rawIATdata[rawIATdata$block >= 6 &
             rawIATdata$ID <= 5, "latency"] <-
  rawIATdata[rawIATdata$block >= 6 &
             rawIATdata$ID <= 5, "latency"] + 100

head(rawIATdata)

#### pretreat IAT data using function Pretreatment ####
IATdata <- Pretreatment(rawIATdata,
                           label_subject = "ID",
                         label_latency = "latency",
                         label_accuracy = "correct",
                         label_block = "block",
                         block_pair1 = c(3, 4),
                         block_pair2 = c(6, 7),
                         label_praccrit = "block",
                         block_prac = c(3, 6),
                         block_crit = c(4, 7))
# data are now in the correct format
head(IATdata)
```

`RobustScores` *Compute the Robust IAT scores*

Description

This is the main function of the package. It allows to compute many variants of the robust IAT scores all with a single command.

Usage

```
RobustScores(IATdata,
P1 = c("none", "fxtrim", "fxwins", "trim10", "wins10", "inve10"),
P2 = c("ignore", "exclude", "recode", "separate", "recode600"),
P3 = c("dscore", "gscore", "wpr90", "minid", "minid_t10", "minid_w10",
"minid_i10"),
P4 = c("nodist", "dist"), maxMemory = 1000,
verbose = TRUE,
autoremove = TRUE)

D2(IATdata, ...)
D5(IATdata, ...)
D6(IATdata, ...)
D2SWND(IATdata, ...)
D5SWND(IATdata, ...)
D6SWND(IATdata, ...)
```

Arguments

`IATdata` a dataframe with the following columns:

- `subject`: (factor or coercible to factor). Univocally identifies a participant.
- `correct`: (logical). has value `TRUE` or `1` if the trial was answered correctly, `FALSE` or `0` otherwise.
- `latency`: (numeric). Response latency, in ms.
- `blockcode`: (factor or string). Can assume only two values, `"pair1"` and `"pair2"`. `"pair1"` is for one critical block and `"pair2"` is the other critical block.
- `praccrit`. (factor, optional). Can assume only two values, `"prac"` is for practice combined categorization block and `"crit"` is for critical combined categorization block. In a IAT with 60 trials for each double categorization block, the first 20 are sometimes administered as practice block, the other 40 as critical.

`P1` (Vector of strings). Determines how the latencies are treated for computing the scores. Can include one or more of the following strings. It is worth noticing that latencies > 10s are excluded by default, independent of P1.

1. `"none"`: Do nothing.
2. `"fxtrim"`: Trim values < 400ms
3. "fxwins": Values < 300ms assume the value 300ms and values > 3000ms assume the value 3000ms
4. `"trim10"`: 10% trimming
5. `"wins10"`: 10% winsorizing
6. `"inve10"`: 10% inverse trimming (i.e., trim central values)

`P2` (Vector of strings). Determines how the error latencies are treated. Can include one or more of the following strings.

1. "ignore": Disregard the correct-error distinction, treat all the latencies as if they were correct latencies.
2. "exclude": Remove error latencies and consider only the correct ones.
3. "recode": Recode the error latencies with the M+2SD of correct latencies. In the computation of the M and of the SD, all correct latencies are considered that are < 10s.
4. "separate": Apply parameter P1 separately for correct and error latencies. Notice that for parameter 1 equal to "none", "fxtrim", and "fxwins", if P4 = "ignore" and P4 = "separate", the result is the same.)
5. "recode600": Recode the error latencies with the the mean of correct latencies + 600ms. In the computation of the Mean, all correct latencies are considered that are < 10s.

P3 The algorithm for computing the Dscores. Can include one or more of the following strings.

1. "dscore". Compute the Dscores as M pair2 - Mpair1 / pooled SD.
2. "gscore". Compute the Gscores, as shown in Nosek, Bar-Anan, Sriram, & Greenwald (2013).
3. "wpr90". Compute the scores based on the worst-performance-rule, which are the same as the Dscores, but instead of the mean, the 90th percentile is used in the numerator.
4. minid. Compute the minidifferences, i.e., the differences between any latency in pair2 and any latency in pair1. Then compute the IAT scores as the Mean of the minidifferences, divided by their SD.
5. minid_t10. Compute the 10% trimmed minidifferences, which are identical to the minidiffernces, but instead of the mean, the 10% trimmed mean is used.
6. "minid_w10" Compute the 10% winsorized minidifferences, which are as the minidifferences, but instead of the mean, the 10% winsorized mean is used.
7. "minid_i10" Compute the 10% inverse_trimmed minidifferences, which are as the minidifferences, but instead of the mean, the 10% inverse trimmed mean is used.

P4 Distinguish the practice and the critical blocks, as specified by column praccrit in the IATdata, or do not.

1. "nodist" no distinction between practice and critical blocks. no distinction is made between practice and critical blocks and the IAT scores are computed using all trials together.
2. "dist" compute the IAT scores as the average IAT score computed. the scores are computed on practice and critical blocks separately: the total score is then computedc as the average of the two IAT scores.

maxMemory In computing the minidifferences, a very large dataframe is required. maxMemory specifies the maximum size of this dataframe, in MB. This limit is respected by "slicing" the dataset and computing the scores separately for many subsets of participants. This can slow the computation a bit, but prevents RAM overflows.

verbose if TRUE, Print the time at which several operations are performed.

`autoremove` if `TRUE` (the default), participants with less than 3 correct responses with latency between 400ms and 10s in each block are excluded from the analyses. Disabling this option can result in computing some variants of IAT scores on too few trials and it can lead to errors and missing values. Change this parameter to `FALSE` only if you know what you are doing.

`...` Additional arguments for `RobustScores`.

Details

A precise description of the parameters can be found in Richetin et al. (2015, Table 1). The procedure for computing the scores is the following.

1. First parameter P4 is applied: for `"nodist"` the whole dataset is given as input, for `"dist"` the dataset is first split in two parts according to column `praccrit` and then given in input.
2. Second, the parameter P1 and P2 are applied: correct and error latencies are treated for each combinations of P1 and P2 and a new column is internally created.
3. Third, parameter P3 is applied. On each and every vector of latencies defined by a combination of P1 and P2, the IAT scores are computed using all the methods specified in P3.
4. Finally, for P4 = `"dist"`, the scores computed i the practice and critical blocks are averaged.

Functions `D2`, `D5`, and `D6` are simple wrappers around RobustScores that allow computing the D2, D5, and D6 scores shown in Greenwald et al. (2003). Similarly, `D2SWND`, `D5SWND`, and `D6SWND` allow computing the same D2, D5, and D6 scores with the improvements proposed by Richetin et al. (2015): use of statistical winsorizing (SW) and no distinction (ND) between practice and critical blocks.

Value

A dataframe with as many columns as subjects, and as many rows as the possible combinations of the parameters P1, P2, P3 and P4.

`subject` The identifier of the participant

.

`p1342` The IAT score variants computed. Each number after the p indicates the value of the parameter corresponding to the position. For instance `p1342` indicates that parameter P1 has value 1 (i.e. `"none"`), parameter P2 has value 3, i.e., `recode`, parameter P3 has value 4 (i.e., `"minid"`) and parameter P4 has value 2 (i.e. `"dist"`). This naming convention was adopted to allow to immediately and precisely know what has been done by reading the name of the score.

`...` other columns in the form `pxxxx`.

Author(s)

Giulio Costantini

References

Greenwald, A. G., Nosek, B. A., & Banaji, M. R. (2003). Understanding and using the Implicit Association Test: I. An improved scoring algorithm. Journal of Personality and Social Psychology, 85(2), 197-216. doi:10.1037/0022-3514.85.2.197

Nosek, B. A., Bar-Anan, Y., Sriram, N., & Greenwald, A. G. (2013). Understanding and Using the Brief Implicit Association Test: I. Recommended Scoring Procedures. SSRN Electronic Journal. doi:10.2139/ssrn.2196002

Richetin, J., Costantini, G., Perugini, M., Schonbrodt, F. (in press). Should we stop looking for a better scoring algorithm for handling Implicit Association Test data? Test of the role of errors, extreme latencies treatment, scoring formula, and practice trials on reliability and validity. PLoS ONE.

See Also

`SplitHalf`, `alg2param`

Examples

```
#### generate random IAT data ####
set.seed(1234)
rawIATdata <- data.frame(
  # ID of each participant (N = 10)
  ID = rep(1:10, each = 180),
  # seven-block structure, as in Greenwald, Nosek & Banaji (2003)
  # block 1 = target discrimination (e.g., Bush vs. Gore items)
  # block 2 = attribute discrimination (e.g., Pleasant words vs. unpleasant)
  # block 3 = combined practice (e.g., Bush + pleasant vs. Gore + unpleasant)
  # block 4 = combined critical  (e.g., Bush + pleasant vs. Gore + unpleasant)
  # block 5 = reversed target discrimination (e.g., Gore vs. Bush)
  # block 6 = reversed combined practice (e.g., Gore + pleasant vs. Bush + unpleasant)
  # block 7 = reversed combined critical (e.g., Gore + pleasant vs. Bush + unpleasant)
  block = rep(c(rep(1:3, each = 20),
                rep(4, 40),
                rep(5:6, each = 20),
                rep(7, 40)), 10),
  # expected proportion of errors = 10 percent
  correct = sample(c(0, 1), size = 1800, replace = TRUE, prob = c(.2, .8)),
  # reaction times are generated from a mix of two chi2 distributions,
  # one centered on 550ms and one on 100ms to simulate fast latencies
  latency = round(sample(c(rchisq(1500, df = 1, ncp = 550),
                           rchisq(300, df = 1, ncp = 100)), 1800)))

# add some IAT effect by making trials longer in block 6 and 7
rawIATdata[rawIATdata$block >= 6, "latency"] <-
  rawIATdata[rawIATdata$block >= 6, "latency"] + 100

# add some more effect for subjects 1 to 5
rawIATdata[rawIATdata$block >= 6 &
             rawIATdata$ID <= 5, "latency"] <-
```

```
  rawIATdata[rawIATdata$block >= 6 &
             rawIATdata$ID <= 5, "latency"] + 100

#### pretreat IAT data using function Pretreatment ####
IATdata <- Pretreatment(rawIATdata,
                            label_subject = "ID",
                          label_latency = "latency",
                          label_accuracy = "correct",
                          label_block = "block",
                          block_pair1 = c(3, 4),
                          block_pair2 = c(6, 7),
                          label_praccrit = "block",
                          block_prac = c(3, 6),
                          block_crit = c(4, 7))


#### Compute Greenwald et al.'s (2003, Table 3) D2, D5, and D6 measures ####
# All scores are computed both with the RobustScores and with
# the wrappers D2, D5, and D6. Results are identical

# D2 scores
D2(IATdata, verbose = FALSE)
RobustScores(IATdata = IATdata,
             P1 = "fxtrim",
             P2 = "ignore",
             P3 = "dscore",
             P4 = "dist",
             verbose = FALSE)

# D5 scores
D5(IATdata, verbose = FALSE)
RobustScores(IATdata = IATdata,
             P1 = "fxtrim",
             P2 = "recode",
             P3 = "dscore",
             P4 = "dist",
             verbose = FALSE)

# D6 scores
D6(IATdata, verbose = FALSE)
RobustScores(IATdata = IATdata,
             P1 = "fxtrim",
             P2 = "recode600",
             P3 = "dscore",
             P4 = "dist",
             verbose = FALSE)

#### Compute D scores with improvements by Richetin et al. (2015, p. 20) ####
# "In this perspective, we examined whether the D2 for built-in penalty and the
# D5 and D6 for no built-in penalty could benefit from the inclusion of two
# elements that stand out from the results. Within their respective parameter,
# the Statistical Winsorizing as a treatment for extreme latencies and No
# distinction between practice and test trials when computing the difference
```

```
# between the two critical blocks seem to lead to the best performances".

# All scores are computed both with the RobustScores and with
# the wrappers D2SWND, D5SWND, and D6SWND. Results are identical


# D2SWND scores
D2SWND(IATdata, verbose = FALSE)
RobustScores(IATdata = IATdata,
             P1 = "wins10",
             P2 = "ignore",
             P3 = "dscore",
             P4 = "nodist",
             verbose = FALSE)

# D5_SWND scores
D5SWND(IATdata, verbose = FALSE)
RobustScores(IATdata = IATdata,
             P1 = "wins10",
             P2 = "recode",
             P3 = "dscore",
             P4 = "nodist",
             verbose = FALSE)

# D6_SWND scores
D6SWND(IATdata, verbose = FALSE)
RobustScores(IATdata = IATdata,
             P1 = "wins10",
             P2 = "recode600",
             P3 = "dscore",
             P4 = "nodist",
             verbose = FALSE)



  #### Compute all 421 combinations of IAT scores ####
  # 421 are the combinations given by parameters P1, P2, P3, and P4. For
  # details, see Richetin et al. (2015)
  allIATscores <- RobustScores(IATdata = IATdata)
```

`SplitHalf` *Split half reliability*

Description

Compute split half reliability for the algorithms defined by all the combinations of parameters P1, P2, P3, and P4.

Usage

```
SplitHalf(IATdata, ...)
SplitHalf.D2(IATdata, ...)
SplitHalf.D5(IATdata, ...)
SplitHalf.D6(IATdata, ...)
SplitHalf.D2SWND(IATdata, ...)
SplitHalf.D5SWND(IATdata, ...)
SplitHalf.D6SWND(IATdata, ...)
```

Arguments

IATdata	same as RobustScores
...	other parameters to be passed to RobustScores

Details

The split-half reliability is computed by splitting the dataframe IATdata in two halves and then calling function RobustScores Functions SplitHalf.D2 etc. are wrappers that allow computing reliability for some common types of scores. See RobustScores.

Value

A vector of split-half reliabilities.

Author(s)

Giulio Costantini

See Also

RobustScores, alg2param

Examples

```
#### generate random IAT data ####
set.seed(1234)
rawIATdata <- data.frame(
  # ID of each participant (N = 10)
  ID = rep(1:10, each = 180),
  # seven-block structure, as in Greenwald, Nosek & Banaji (2003)
  # block 1 = target discrimination (e.g., Bush vs. Gore items)
  # block 2 = attribute discrimination (e.g., Pleasant words vs. unpleasant)
  # block 3 = combined practice (e.g., Bush + pleasant vs. Gore + unpleasant)
  # block 4 = combined critical  (e.g., Bush + pleasant vs. Gore + unpleasant)
  # block 5 = reversed target discrimination (e.g., Gore vs. Bush)
  # block 6 = reversed combined practice (e.g., Gore + pleasant vs. Bush + unpleasant)
  # block 7 = reversed combined critical (e.g., Gore + pleasant vs. Bush + unpleasant)
  block = rep(c(rep(1:3, each = 20),
                rep(4, 40),
                rep(5:6, each = 20),
```

```
                rep(7, 40)), 10),
 # expected proportion of errors = 10 percent
  correct = sample(c(0, 1), size = 1800, replace = TRUE, prob = c(.2, .8)),
  # reaction times are generated from a mix of two chi2 distributions,
  # one centered on 550ms and one on 100ms to simulate fast latencies
  latency = round(sample(c(rchisq(1500, df = 1, ncp = 550),
                           rchisq(300, df = 1, ncp = 100)), 1800)))

# add some IAT effect by making trials longer in block 6 and 7
rawIATdata[rawIATdata$block >= 6, "latency"] <-
  rawIATdata[rawIATdata$block >= 6, "latency"] + 100

# add some more effect for subjects 1 to 5
rawIATdata[rawIATdata$block >= 6 &
             rawIATdata$ID <= 5, "latency"] <-
  rawIATdata[rawIATdata$block >= 6 &
             rawIATdata$ID <= 5, "latency"] + 100


#### pretreat IAT data using function Pretreatment ####
IATdata <- Pretreatment(rawIATdata,
                           label_subject = "ID",
                         label_latency = "latency",
                         label_accuracy = "correct",
                         label_block = "block",
                         block_pair1 = c(3, 4),
                         block_pair2 = c(6, 7),
                         label_praccrit = "block",
                         block_prac = c(3, 6),
                         block_crit = c(4, 7))


#### Compute reliability for Greenwald et al.'s (2003) D2, D5, and D6 ####
# All scores are computed both with the SplitHalf and with
# the wrappers SplitHalf.D2, SplitHalf.D5, and SplitHalf.D6.


# D2 scores
SplitHalf.D2(IATdata, verbose = FALSE)
SplitHalf(IATdata = IATdata,
             P1 = "fxtrim",
             P2 = "ignore",
             P3 = "dscore",
             P4 = "dist",
             verbose = FALSE)

# D5 scores
SplitHalf.D5(IATdata,
             verbose = FALSE)
SplitHalf(IATdata = IATdata,
             P1 = "fxtrim",
             P2 = "recode",
             P3 = "dscore",
```

```
              P4 = "dist",
              verbose = FALSE)

# D6 scores
SplitHalf.D6(IATdata, verbose = FALSE)
SplitHalf(IATdata = IATdata,
              P1 = "fxtrim",
              P2 = "recode600",
              P3 = "dscore",
              P4 = "dist",
              verbose = FALSE)

#### Compute reliability for improved scores by Richetin et al. (2015, p. 20) ####
# All scores are computed both with the SplitHalf and with
# the wrappers SplitHalf.D2SWND, SplitHalf.D5SWND, and SplitHalf.D6SWND.
# Results are identical

# D2SWND scores
SplitHalf.D2SWND(IATdata, verbose = FALSE)
SplitHalf(IATdata = IATdata,
              P1 = "wins10",
              P2 = "ignore",
              P3 = "dscore",
              P4 = "nodist",
              verbose = FALSE)

# D5_SWND scores
SplitHalf.D5SWND(IATdata, verbose = FALSE)
SplitHalf(IATdata = IATdata,
              P1 = "wins10",
              P2 = "recode",
              P3 = "dscore",
              P4 = "nodist",
              verbose = FALSE)

# D6_SWND scores
SplitHalf.D6SWND(IATdata, verbose = FALSE)
SplitHalf(IATdata = IATdata,
              P1 = "wins10",
              P2 = "recode600",
              P3 = "dscore",
              P4 = "nodist",
              verbose = FALSE)
```

`TestRetest` *Test-retest reliability*

Description

Compute test-retest reliability for IAT with 2 observations for each subject

Usage

```
TestRetest(IATdata, ...)
TestRetest.D2(IATdata, ...)
TestRetest.D5(IATdata, ...)
TestRetest.D6(IATdata, ...)
TestRetest.D2SWND(IATdata, ...)
TestRetest.D5SWND(IATdata, ...)
TestRetest.D6SWND(IATdata, ...)
```

Arguments

IATdata	same as RobustScores, but with the additional column "session". session distinguishes the trials of the first session and those of the second session. It is typically numerical, having value 1 for the first session and 2 for the second. Functions TestRetest.D2 etc. are wrappers that allow computing reliability for some common types of scores. See RobustScores.
...	other parameters to be passed to RobustScores

Details

It computes the scores for the test and for the retest using RobustScores, the output is just the correlation among the scores in the two sessions.

Value

algorithm	The name of the algorithm, see RobustScores for the convention adopted for naming the algorithms
testretest	The test-retest reliability for each algorithm

Author(s)

Giulio Costantini

See Also

RobustScores

Examples

```
#### generate random IAT data ####
set.seed(1234)
rawIATdata <- data.frame(
  # ID of each participant (N = 10)
  ID = rep(1:10, each = 180),
  # seven-block structure, as in Greenwald, Nosek & Banaji (2003)
  # block 1 = target discrimination (e.g., Bush vs. Gore items)
  # block 2 = attribute discrimination (e.g., Pleasant words vs. unpleasant)
  # block 3 = combined practice (e.g., Bush + pleasant vs. Gore + unpleasant)
  # block 4 = combined critical  (e.g., Bush + pleasant vs. Gore + unpleasant)
  # block 5 = reversed target discrimination (e.g., Gore vs. Bush)
```

```
  # block 6 = reversed combined practice (e.g., Gore + pleasant vs. Bush + unpleasant)
  # block 7 = reversed combined critical (e.g., Gore + pleasant vs. Bush + unpleasant)
  block = rep(c(rep(1:3, each = 20),
                rep(4, 40),
                rep(5:6, each = 20),
                rep(7, 40)), 10),
 # expected proportion of errors = 10 percent
  correct = sample(c(0, 1), size = 1800, replace = TRUE, prob = c(.2, .8)),
  # reaction times are generated from a mix of two chi2 distributions,
  # one centered on 550ms and one on 100ms to simulate fast latencies
  latency = round(sample(c(rchisq(1500, df = 1, ncp = 550),
                           rchisq(300, df = 1, ncp = 100)), 1800)))

# add some IAT effect by making trials longer in block 6 and 7
rawIATdata[rawIATdata$block >= 6, "latency"] <-
  rawIATdata[rawIATdata$block >= 6, "latency"] + 100

# add some more effect for subjects 1 to 5
rawIATdata[rawIATdata$block >= 6 &
             rawIATdata$ID <= 5, "latency"] <-
  rawIATdata[rawIATdata$block >= 6 &
             rawIATdata$ID <= 5, "latency"] + 100

#### pretreat IAT data using function Pretreatment ####
IATdata <- Pretreatment(rawIATdata,
                           label_subject = "ID",
                         label_latency = "latency",
                         label_accuracy = "correct",
                         label_block = "block",
                         block_pair1 = c(3, 4),
                         block_pair2 = c(6, 7),
                         label_praccrit = "block",
                         block_prac = c(3, 6),
                         block_crit = c(4, 7))

# Add a column representing the session in IATdata
IATdata$session <- rep(c(1,2), nrow(IATdata)/2)


#### Compute reliability for Greenwald et al.'s (2003) D2, D5, and D6 ####
# All scores are computed both with the TestRetest and with
# the wrappers TestRetest.D2, TestRetest.D5, and TestRetest.D6.


# D2 scores
TestRetest.D2(IATdata, verbose = FALSE)
TestRetest(IATdata = IATdata,
             P1 = "fxtrim",
             P2 = "ignore",
             P3 = "dscore",
             P4 = "dist",
             verbose = FALSE)
```

```
# D5 scores
TestRetest.D5(IATdata,
              verbose = FALSE)
TestRetest(IATdata = IATdata,
              P1 = "fxtrim",
              P2 = "recode",
              P3 = "dscore",
              P4 = "dist",
              verbose = FALSE)

# D6 scores
TestRetest.D6(IATdata, verbose = FALSE)
TestRetest(IATdata = IATdata,
              P1 = "fxtrim",
              P2 = "recode600",
              P3 = "dscore",
              P4 = "dist",
              verbose = FALSE)

#### Compute reliability for improved scores by Richetin et al. (2015, p. 20) ####
# All scores are computed both with the TestRetest and with
# the wrappers TestRetest.D2SWND, TestRetest.D5SWND, and TestRetest.D6SWND.
# Results are identical

# D2SWND scores
TestRetest.D2SWND(IATdata, verbose = FALSE)
TestRetest(IATdata = IATdata,
              P1 = "wins10",
              P2 = "ignore",
              P3 = "dscore",
              P4 = "nodist",
              verbose = FALSE)

# D5_SWND scores
TestRetest.D5SWND(IATdata, verbose = FALSE)
TestRetest(IATdata = IATdata,
              P1 = "wins10",
              P2 = "recode",
              P3 = "dscore",
              P4 = "nodist",
              verbose = FALSE)

# D6_SWND scores
TestRetest.D6SWND(IATdata, verbose = FALSE)
TestRetest(IATdata = IATdata,
              P1 = "wins10",
              P2 = "recode600",
              P3 = "dscore",
              P4 = "nodist",
              verbose = FALSE)
```

Tgraph *Layout* qgraph *for multiple comparisons by package* nparcomp

Description

Implements the T-graph layout proposed by Vasilescu et al. (2014), using the robust nonparametric contrasts proposed by Konietschke et al. (2012).

Usage

```
Tgraph(mcmp, alpha = 0.05, horizorder = NULL)
```

Arguments

mcmp	The output of a robust post-hoc, as obtained with function mcmp from package nparcomp
alpha	The alpha level, by convention = .05. Effects with p.values lower than alpha are represented as arrows in the network layout.
horizorder	Optional, vector of strings. While the vertical order of the variables in the Tgraph is determined by the multiple comparisons, the horizontal ordering is not. If specified, parameter horizorder allows to determine the horizontal order. It must be a vector with the names of the variables in the preferred horizontal order.

Details

A T-graph is a simple graphical representation of a series of pairwise comparison proposed by Vasilescu et al. (2014). The nodes of the graph represent the levels of the factor, the arrows represent their pairwise comparisons. An arrow points from one option to another if the dependent variable is significantly higher for the first level compared to the second level of the factor. The robust contrasts defined by Konietschke et al. (2012) have the transitive property, therefore if an option X outperforms another option Y and Y outperforms Z, this implies that X outperforms Z. For sake of a clear graphical representation we followed Vasilescu et al. and omitted the direct edges when two nodes could be connected using an indirect path travelling through other nodes.

Value

wmat	The weights matrix, for each pair of options the weights represent the value of the estimated relative effect, see mcmp. A value is present in wmat only if the associated p.value is less than alpha and it is zero otherwise.
amat	The adjacency matrix, for each pair of options, it has value 1 if an edge is present in wmat and 0 otherwise. This should be given as the main input to link{qgraph}
layout	The layout to give in input to qgraph's parameter layout

Author(s)

Giulio Costantini

References

Epskamp, S., Cramer, A. O. J., Waldorp, L. J., Schmittmann, V. D., & Borsboom, D. (2012). qgraph: network visualizations of relationships in psychometric data. Journal of Statistical Software, 48(4).

Konietschke, F., Hothorn, L. a., & Brunner, E. (2012). Rank-based multiple test procedures and simultaneous confidence intervals. Electronic Journal of Statistics, 6, 738-759. doi:10.1214/12-EJS691

Vasilescu, B., Serebrenik, A., Goeminne, M., & Mens, T. (2014). On the variation and specialization of workload-A case study of the Gnome ecosystem community. Empirical Software Engineering, 19, 955-1008. doi:10.1007/s10664-013-9244-1

Richetin, J., Costantini, G., Perugini, M., Schonbrodt, F. (in press). Should we stop looking for a better scoring algorithm for handling Implicit Association Test data? Test of the role of errors, extreme latencies treatment, scoring formula, and practice trials on reliability and validity. PLoS ONE.

Examples

```
library(nparcomp)
library(qgraph)

dat <- data.frame(matrix(nrow = 300, ncol = 0))

dat$DV <- c(rnorm(100, 1, 1),
       rnorm(100, 0, 1),
       rnorm(100, 0, 1))

dat$IV <- c(rep("A", 100),
        rep("B", 100),
        rep("D", 100))

mcmp <- mctp(formula = DV~IV, data = dat, type = "Tukey")
tg <- Tgraph(mcmp)
qgraph(tg$amat, layout = tg$layout)

tg2 <- Tgraph(mcmp,  horizorder = c("A", "D", "B"))
qgraph(tg2$amat, layout = tg2$layout)
```