Package 'stuart'

September 17, 2018

Type Package

Title Subtests Using Algorithmic Rummaging Techniques

Version 0.7.3

Date 2018-09-11

Description Construct subtests from a pool of items by using ant-colony-optimization, genetic algorithms, brute force, or random sampling. Schultze (2017) <doi:10.17169/refubium-622>.

License GPL-3

Depends R (>= 3.5.0)

Imports stats, utils

Suggests parallel, lavaan (>= 0.5.18), MplusAutomation (>= 0.7-2), graphics

RoxygenNote 6.0.1

Encoding UTF-8

NeedsCompilation no

Author Martin Schultze [aut, cre]

Maintainer Martin Schultze <martin.schultze@fu-berlin.de>

Repository CRAN

Date/Publication 2018-09-17 15:30:03 UTC

R topics documented:

`stuart-package` *STUART: Subtests Using Algorithmic Rummaging Techniques*

Description

The STUART-Package automates the generation of subtests from a given set of items within the confines of confirmatory factor analysis.

Functionality

Using this package subtests can be generated in four different ways: using a pseudo-random approach rooted in Ant-Colony-Optimization via the `mmas`-function, using a simple genetic algorithm via the `gene`-function, using a brute-force approach via the aptly named `bruteforce`-function, or by random chance, using the `randomsamples`-function.

Addtionally, there are some convenience functions which are more or less useful. The `combinations`-function can be used to determine the number of possible subtests to inform a decision on which selection approach to use. The `crossvalidate`-function can be used to evaluate the quality of a selection in a different (sub-)sample. To add to this functionality, the `holdout`-function randomly splits the data into a calibration and a validation sample. The `heuristics`-function can be used to extract the formatting of heuristic matrices which can be provided to the `mmas`-function.

The package also provides two datasets: the `sups` dataset and the `fairplayer` dataset.

Author(s)

Maintainer: Martin Schultze <martin.schultze@fu-berlin.de>

`bruteforce` *Subtest construction using a brute-force approach*

Description

Construct subtests from a given pool of items using a brute-force approach (i.e. by estimating all possible combinations).

Usage

```
bruteforce(data, factor.structure, capacity = NULL,
  item.invariance = "congeneric", repeated.measures = NULL,
  long.invariance = "strict", mtmm = NULL, mtmm.invariance = "configural",
  grouping = NULL, group.invariance = "strict", auxiliary = NULL,
  use.order = FALSE, software = "lavaan", cores = NULL,
  objective = objective.preset, ignore.errors = FALSE,
  analysis.options = NULL, suppress.model = FALSE,
  request.override = 10000, filename = NULL)
```

Arguments

`data`	A data.frame containing all relevant data.
`factor.structure`	A list linking factors to items. The names of the list elements correspond to the factor names. Each list element must contain a character-vector of item names that are indicators of this factor.
`capacity`	A list containing the number of items per subtest. This must be in the same order as the `factor.structure` provided. If a single number, it is applied to all subtests. If `NULL` all items are evenly distributed among the subtests.
`item.invariance`	A character vector of length 1 or the same length as `factor.structure` containing the desired invariance levels between items pertaining to the same subtest. Currently there are five options: 'congeneric', 'ess.equivalent', 'ess.parallel', 'equivalent', and 'parallel', the first being the default.
`repeated.measures`	A list linking factors that are repeated measures of each other. Repeated factors must be in one element of the list - other sets of factors in other elements of the list. When this is `NULL` (the default) a cross-sectional model is estimated.
`long.invariance`	A character vector of length 1 or the same length as `repeated.measures` containing the longitudinal invariance level of repeated items. Currently there are four options: 'configural', 'weak', 'strong', and 'strict'. Defaults to 'strict'. When `repeated.measures=NULL` this argument is ignored.
`mtmm`	A list linking factors that are measurements of the same construct with different methods. Measurements of the same construct must be in one element of the list - other sets of methods in other elements of the list. When this is `NULL` (the default) a single method model is estimated.
`mtmm.invariance`	A character vector of length 1 or the same length as `mtmm` containing the invariance level of MTMM items. Currently there are five options: 'none', 'configural', 'weak', 'strong', and 'strict'. Defaults to 'configural'. With 'none' differing items are allowed for different methods. When `mtmm=NULL` this argument is ignored.
`grouping`	The name of the grouping variable. The grouping variable must be part of `data` provided and must be a numeric variable.

`group.invariance`
: A single value describing the assumed invariance of items across groups. Currently there are four options: 'configural', 'weak', 'strong', and 'strict'. Defaults to 'strict'. When `grouping=NULL` this argument is ignored.

`auxiliary`
: The names of auxiliary variables in `data`. These can be used in additional modeling steps that may be provided in `analysis.options$model`.

`use.order`
: A logical indicating whether or not to take the selection order of the items into account. Defaults to `FALSE`.

`software`
: The name of the estimation software. Can currently be 'lavaan' (the default), 'Mplus', or 'Mplus Demo'. Each option requires the software to be installed.

`cores`
: The number of cores to be used in parallel processing. If `NULL` (the default) the result of `detectCores` will be used. On Unix-y machines parallel processing is implemented via `mclapply`, on Windows machines it is realized via `parLapply`.

`objective`
: A function that converts the results of model estimation into a pheromone. See `mmas` for details.

`ignore.errors`
: A logical indicating whether or not to ignore estimation problems (such as non positive-definite latent covariance matrices). Defaults to `FALSE`.

`analysis.options`
: A list additional arguments to be passed to the estimation software. The names of list elements must correspond to the arguments changed in the respective estimation software. E.g. `analysis.options$model` can contain additional modeling commands - such as regressions on auxiliary variables.

`suppress.model`
: A logical indicating whether to suppress the default model generation. If `TRUE` a model must be provided in `analysis.options$model`.

`request.override`
: The maximum number of combinations for which the estimation is performed immediately, without an additional override request.

`filename`
: The stem of the filenames used to save inputs, outputs, and data files when `software='Mplus'`. This may include the file path. When `NULL` (the default) files will be saved to the temporary directory, which is deleted when the R session is ended.

Value

Returns an object of the class `stuartOutput` for which specific `summary` and `plot` methods are available. The results are a list.

`call`
: The called function.

`software`
: The software used to fit the CFA models.

`parameters`
: A list of the ACO parameters used.

`analysis.options`
: A list of the additional arguments passed to the estimation software.

`timer`
: An object of the class `proc_time` which contains the time used for the analysis.

`log`
: A `data.frame` containing the estimation history.

`solution`
: `NULL`

pheromones	NULL
subtests	A list containing the names of the selected items and their respective subtests.
final	The results of the estimation of the global-best solution.

Author(s)

Martin Schultze

See Also

mmas, gene, randomsamples, combinations

Examples

```
# Bruteforce selection in a minimal example
# selecting 3 of 5 items
# requires lavaan
data(fairplayer)
fs <- list(ra = names(fairplayer)[53:57])
sel <- bruteforce(fairplayer, fs, 3,
  cores = 1)  # number of cores set to 1
summary(sel)  # Fit is perfect because of just-identified model
```

---

combinations *Compute the number of possible subtest combinations*

Description

Used to compute the number of possible subtest constellations prior to performing item selection.

Usage

```
combinations(data, factor.structure, capacity = NULL,
  repeated.measures = NULL, mtmm = NULL, use.order = FALSE, ...)
```

Arguments

data	A data.frame containing all relevant data.
factor.structure	A list linking factors to items. The names of the list elements correspond to the factor names. Each list element must contain a character-vector of item names that are indicators of this factor.
capacity	A list containing the number of items per subtest. This must be in the same order as the factor.structure provided. If a single number, it is applied to all subtests. If NULL all items are evenly distributed among the subtests.

repeated.measures
: A list linking factors that are repeated measures of each other. Repeated factors must be in one element of the list - other sets of factors in other elements of the list. When this is NULL (the default) a cross-sectional model is estimated.

mtmm
: A list linking factors that are measurements of the same construct with different methods. Measurements of the same construct must be in one element of the list - other sets of methods in other elements of the list. When this is NULL (the default) a single method model is estimated.

use.order
: A logical indicating whether or not to take the selection order of the items into account. Defaults to FALSE.

...
: Other arguments normally provided to mmas, which will be ignored.

Value

Returns the number of possible subtest constellations.

Author(s)

Martin Schultze

See Also

bruteforce, mmas, gene

Examples

```
# Determine number of combinations in a simple situation
data(fairplayer)
fs <- list(si = names(fairplayer)[83:92])
combinations(fairplayer, fs, 4)

# Number of combinations with repeated measures
data(fairplayer)
fs <- list(si1 = names(fairplayer)[83:92],
  si2 = names(fairplayer)[93:102],
  si3 = names(fairplayer)[103:112])
repe <- list(si = c('si1', 'si2', 'si3'))
combinations(fairplayer, fs, 4, repeated.measures = repe)
```

---

crossvalidate *Cross-Validate a Measurement Model*

Description

Cross-validate a measurement model obtained from STUART.

Usage

```
crossvalidate(selection, old.data, new.data, invariance = "configural",
  objective = NULL, filename = NULL)
```

Arguments

selection	An object of class stuartOutput.
old.data	A data.frame of the calibration sample.
new.data	A data.frame of the validation sample.
invariance	The invariance between the calibration and the validation sample. Can be one of 'configural', 'weak', 'strong', 'strict', or 'full', with the first being the default. Currently 'full' is only functional when using Mplus.
objective	A function that converts the results of model estimation into a pheromone. If none is provided the default function fitness is used. This can be examined with body(stuart:::fitness).
filename	The stem of the filenames used to save inputs, outputs, and data files when software='Mplus'. This may include the file path. When NULL (the default) files will be saved to the temporary directory, which is deleted when the R session is ended.

Value

Returns a list containing the data.frame comparison and an object containing the model results of the validation sample.

comparison	A data.frame with 2 observations. The first observation shows the components of the objective function for the final model in the calibration sample. The second observation those of the model for the validation sample. Which variables are returned depends on the setting of objective.
validation	When using lavaan for estimation, an object of class lavaan containing the model results fit to the validation sample. When using Mplus for estimation, a character vector containing the Mplus output for the validation sample.

Author(s)

Martin Schultze

See Also

holdout, mmas, bruteforce

Examples

```
# Split data into two halves
data(fairplayer)
half1 <- fairplayer[1:72,]
half2 <- fairplayer[73:143,]
```

```
# Simple example from bruteforce
fs <- list(ra = names(fairplayer)[53:57])
sel <- bruteforce(half1, fs, 3,
  cores = 1)  # number of cores set to 1

# Validation
crossvalidate(sel, half1, half2,
  invariance = 'strong')  # assuming equality of loadings and intercepts

# Using the 'holdout' function for data split
data(fairplayer)
split <- holdout(fairplayer, seed = 55635)

# Simple example from bruteforce
fs <- list(ra = names(fairplayer)[53:57])
sel <- bruteforce(split, fs, 3,
  cores = 1)  # number of cores set to 1

# Validation
crossvalidate(sel, split,
  invariance = 'weak')  # assuming equality of loadings
```

`fairplayer` *MTMM fairplayer Intervention Data (2009)*

Description

Self- and teacher-reported empathy (8 item scale), relational aggression (5 item scale), and social intelligence (10 item scale) at three different occasions.

Usage

```
fairplayer
```

Format

A data frame with 143 observations on 142 variables. The variable names consist of an initial letter indicating the source (s: self-report, t: teacher-report), two letters indicating the construct (EM: empathy, RA: relational aggression, SI: social intelligence), a number indicating the item number on the scale, and a "t" followed by a number indicating the measurement occasion.

Source

Bull, H., Schultze, M., Scheithauer, H. (2009) School-based prevention of bullying and relational aggression: The fairplayer.manual. European Journal of Developmental Science, 3:313-317.

Schultze, M. (2012). Evaluating What The Crowd Says. A longitudinal structural equation model for exchangeable and structurally different methods for evaluating interventions. Unpublished Diploma Thesis.

gene *Subtest construction using a simple genetic algorithm*

Description

Construct subtests from a given pool of items using a simple genetic algorithm. Allows for multiple constructs, occasions, and groups.

Usage

```
gene(data, factor.structure, capacity = NULL, item.weights = NULL,
  item.invariance = "congeneric", repeated.measures = NULL,
  long.invariance = "strict", mtmm = NULL, mtmm.invariance = "configural",
  grouping = NULL, group.invariance = "strict", auxiliary = NULL,
  use.order = FALSE, software = "lavaan", cores = NULL,
  objective = objective.preset, ignore.errors = FALSE, generations = 128,
  individuals = 64, elitism = 1/individuals, reproduction = 0.5,
  mutation = 0.1, mating.index = 1, mating.size = 0.25,
  mating.criterion = "fitness", tolerance = 1e-04,
  analysis.options = NULL, suppress.model = FALSE, seed = NULL,
  filename = NULL)
```

Arguments

`data` A data.frame containing all relevant data.

`factor.structure`
A list linking factors to items. The names of the list elements correspond to the factor names. Each list element must contain a character-vector of item names that are indicators of this factor.

`capacity` A list containing the number of items per subtest. This must be in the same order as the `factor.structure` provided. If a single number, it is applied to all subtests. If `NULL` all items are evenly distributed among the subtests.

`item.weights` A placeholder. Currently all weights are assumed to be one.

`item.invariance`
A character vector of length 1 or the same length as `factor.structure` containing the desired invariance levels between items pertaining to the same subtest. Currently there are five options: 'congeneric', 'ess.equivalent', 'ess.parallel', 'equivalent', and 'parallel', the first being the default.

`repeated.measures`
A list linking factors that are repeated measures of each other. Repeated factors must be in one element of the list - other sets of factors in other elements of the list. When this is `NULL` (the default) a cross-sectional model is estimated.

`long.invariance`
A character vector of length 1 or the same length as `repeated.measures` containing the longitudinal invariance level of repeated items. Currently there are four options: 'configural', 'weak', 'strong', and 'strict'. Defaults to 'strict'. When `repeated.measures=NULL` this argument is ignored.

`mtmm` A list linking factors that are measurements of the same construct with different methods. Measurements of the same construct must be in one element of the list - other sets of methods in other elements of the list. When this is `NULL` (the default) a single method model is estimated.

`mtmm.invariance`
A character vector of length 1 or the same length as `mtmm` containing the invariance level of MTMM items. Currently there are five options: 'none', 'configural', 'weak', 'strong', and 'strict'. Defaults to 'configural'. With 'none' differing items are allowed for different methods. When `mtmm=NULL` this argument is ignored.

`grouping` The name of the grouping variable. The grouping variable must be part of `data` provided and must be a numeric variable.

`group.invariance`
A single value describing the assumed invariance of items across groups. Currently there are four options: 'configural', 'weak', 'strong', and 'strict'. Defaults to 'strict'. When `grouping=NULL` this argument is ignored.

`auxiliary` The names of auxiliary variables in `data`. These can be used in additional modeling steps that may be provided in `analysis.options$model`.

`use.order` A logical indicating whether or not to take the selection order of the items into account. Defaults to `FALSE`.

`software` The name of the estimation software. Can currently be 'lavaan' (the default) or 'Mplus'. Each option requires the software to be installed.

`cores` The number of cores to be used in parallel processing. If `NULL` (the default) the result of `detectCores` will be used. On Unix-y machines parallel processing is implemented via `mclapply`, on Windows machines it is realized via `parLapply`.

`objective` A function that converts the results of model estimation into a pheromone. See 'details' for... details.

`ignore.errors` A logical indicating whether or not to ignore estimation problems (such as non positive-definite latent covariance matrices). Defaults to `FALSE`.

`generations` Maximum number of generations to run. Defaults to 128.

`individuals` The number of individuals per generation. Defaults to 64.

`elitism` The proportion of individuals from the last generation to carry over to the next generation. Defaults to 1/individuals, meaning that the best individual is retained into the next generation.

`reproduction` The proportion of individuals that are allowed to sire offspring. These individuals are selected using fitness proportionate selection. Defaults to .5.

`mutation` The mutation probability. Defaults to .1. See 'details'.

`mating.index` The relative rank of the selected mate within the mating pool. A number bewteen 0, indicating a best-last mating, and 1 (the default), indicating a best-first mating. See 'details'.

`mating.size` The proportion of potential mates sampled from the pool of reproducers for each selected individual. Defaults to .25. See 'details'.

`mating.criterion`
The criterion by which to select mates. Can be either 'similarity' or 'fitness' (the default). See 'details'.

`tolerance` The tolerance for deteriming convergence. Defaults to .0001. See 'details'.

`analysis.options`
A list additional arguments to be passed to the estimation software. The names of list elements must correspond to the arguments changed in the respective estimation software. E.g. `analysis.options$model` can contain additional modeling commands - such as regressions on auxiliary variables.

`suppress.model` A logical indicating whether to suppress the default model generation. If `TRUE` a model must be provided in `analysis.options$model`.

`seed` A random seed for the generation of random samples. See `Random` for more details.

`filename` The stem of the filenames used to save inputs, outputs, and data files when `software='Mplus'`. This may include the file path. When `NULL` (the default) files will be saved to the temporary directory, which is deleted when the R session is ended.

Details

The pheromone function provided via `objective` is used to assess the quality of the solutions. These functions can contain any combination of the fit indices provided by the estimation software. When using Mplus these fit indices are 'rmsea', 'srmr', 'cfi', 'tli', 'chisq' (with 'df' and 'pvalue'), 'aic', 'bic', and 'abic'. With lavaan any fit index provided by `inspect` can be used. Additionally 'crel' provides an aggregate of composite reliabilites, 'rel' provides a vector or a list of reliability coefficients for the latent variables, 'con' provides an aggregate consistency estimate for MTMM analyses, and 'lvcor' provides a list of the latent variable correlation matrices. For more detailed objective functions 'lambda', 'theta', 'psi', and 'alpha' provide the model-implied matrices. Per default a pheromone function using 'crel', 'rmsea', and 'srmr' is used. Please be aware that the `objective` must be a function with the required fit indices as (correctly named) arguments.

The genetic algorithm implemented selects parents in a two-step procedure. First, a fitness proportionate selection is performed to select `inviduals` times `reproduction` viable parents. Then, the non-self-adaptive version of mating proposed by Galán, Mengshoel, and Pinter (2013) is used to perform mating. In contrast to the original article, the `mating.index` and `mating.size` are handled as proportions, not integers. Similarity-based mating is based on the Jaccard Similarity. Mutation is currently always handled as an exchange of the selection state between two items. This results in mutation selecting one item that was not selected prior to mutation and dropping one item selected prior to mutation. Convergence is checked via the variance of the global-best values on the objective function, as proposed by Bhandari, Murthy, and Pal (2012). To avoid false convergence in the early search, the lower of either 10% of the generations or 10 generations must be completed, before convergence is checked. For generalizability over different functions provided to `objective`, these are scaled to the first global-best found.

Value

Returns an object of the class `stuartOutput` for which specific `summary` and `plot` methods are available. The results are a list.

`call` The called function.

`software` The software used to fit the CFA models.

parameters	A list of the parameters used.
analysis.options	A list of the additional arguments passed to the estimation software.
timer	An object of the class proc_time which contains the time used for the analysis.
log	A data.frame containing the optimization history.
solution	A list of matrices with the choices made in the global-best solution.
pheromones	NULL
subtests	A list containing the names of the selected items and their respective subtests.
final	The results of the estimation of the global-best solution.

Author(s)

Martin Schultze

References

Bhandari, D., Murthy, C.A., & Pal, S.K. (2012). Variance as a Stopping Criterion for Genetic Algorithms with Elitist Model. Fundamenta Informaticae, 120, 145-164. doi:10.3233/FI-2012-754

Galán, S.F., Mengshoel, O.J., & Pinter, R. (2013). A novel mating approach for genetic algorithms. Evolutionary Computation, 21(2), 197-229. doi:10.1162/EVCO_a_00067

See Also

bruteforce, mmas, randomsamples

Examples

```
# Genetic selection in a simple situation
# requires lavaan
# number of cores set to 1 in all examples
data(fairplayer)
fs <- list(si = names(fairplayer)[83:92])

# minimal example
sel <- gene(fairplayer, fs, 4,
  generations = 1, individuals = 10,  # minimal runtime, remove for application
  seed = 55635, cores = 1)
summary(sel)


# longitudinal example
data(fairplayer)
fs <- list(si1 = names(fairplayer)[83:92],
  si2 = names(fairplayer)[93:102],
  si3 = names(fairplayer)[103:112])

repe <- list(si = c('si1', 'si2', 'si3'))
```

```
# run to convergence
# switching to best-last mating and 50\% mating size
sel <- gene(fairplayer, fs, 4,
  repeated.measures = repe, long.invariance = 'strong',
  mating.index = 0, mating.size = .5,
  seed = 55635, cores = 1)

# forcing a run through all generations
# by disabling the variance convergence rule
sel <- gene(fairplayer, fs, 4,
  repeated.measures = repe, long.invariance = 'strong',
  tolerance = 0, seed = 55635,
  cores = 1)
```

`heuristics` *Generating heuristics for the use in STUART subtest construction*

Description

Creates uninformative heuristic matrices for the use in `mmas`.

Usage

```
heuristics(data, factor.structure, capacity = NULL,
  repeated.measures = NULL, mtmm = NULL, grouping = NULL,
  localization = "nodes", ...)
```

Arguments

`data`	A data.frame containing all relevant data.
`factor.structure`	A list linking factors to items. The names of the list elements correspond to the factor names. Each list element must contain a character-vector of item names that are indicators of this factor.
`capacity`	A list containing the number of items per subtest. This must be in the same order as the `factor.structure` provided. If a single number, it is applied to all subtests. If `NULL` all items are evenly distributed among the subtests.
`repeated.measures`	A list linking factors that are repeated measures of each other. Repeated factors must be in one element of the list - other sets of factors in other elements of the list. When this is `NULL` (the default) a cross-sectional model is estimated.
`mtmm`	A list linking factors that are measurements of the same construct with different methods. Measurements of the same construct must be in one element of the list - other sets of methods in other elements of the list. When this is `NULL` (the default) a single method model is estimated.

grouping	The name of the grouping variable. The grouping variable must be part of data provided and must be a numeric variable.
localization	Which parameterization to use when depositing pheromones. Can be either 'nodes' (the default) for depositing pheromones on selected nodes or 'arcs' for depositing on selection arcs.
...	Other arguments normally provided to mmas, which will be ignored.

Details

This function generates a list of matrices which can be used as heuristics for all STUART constructions. This is mainly intended to write the structure of the heuristic matrices to an object, change components in line with theoretically derived heuristics and feed them back into mmas via the heuristics argument. The generated heuristics will contain only 1s and 0s, making it no heuristic information. Selection probabilities can be altered by manipulating the contents of the object created by heuristics. Setting a value to 0 will result in prohibiting a certain choice to be made. Please note, that it will lead to unpredictable behavior if the diagonal elements of the matrices produced in the arcs parameterization are set to values other than 0.

Value

Returns a list of the same length as the factor.structure argument provided.

Author(s)

Martin Schultze

See Also

mmas

Examples

```
# heuristics for node localization
data(fairplayer)
fs <- list(si = names(fairplayer)[83:92])

(heu <- heuristics(fairplayer, fs, 4))

# Define anchor-item
heu$si[1] <- 10000
heu

# heuristics for arc localization
data(fairplayer)
fs <- list(si = names(fairplayer)[83:92])

(heu <- heuristics(fairplayer, fs, 4, localization = 'arcs'))

# Define equal selection of odd and even items
heu$si[1:10,] <- c(rep(c(0, 1), 5), rep(c(1, 0), 5))
```

heu

---

holdout *Data selection for holdout validation.*

Description

Split a data.frame into two subsets for holdout validation.

Usage

```
holdout(data, prop = 0.5, grouping = NULL, seed = NULL)
```

Arguments

data	A data.frame.
prop	A single value or vector of proportions of data in calibration sample. Defaults to .5, for an even split.
grouping	Name of the grouping variable. Providing a grouping variable ensures that the provided proportion is selected within each group.
seed	A random seed. See Random for more details.

Value

Returns a list containing two data.frames, called calibrate and validate. The first corresponds to the calibration sample, the second to the validation sample.

Author(s)

Martin Schultze

See Also

crossvalidate

Examples

```
# seeded selection, 25% validation sample
data(fairplayer)
split <- holdout(fairplayer, .75, seed = 55635)
lapply(split, nrow) # check size of samples
```

mmas *Subtest construction using the Max-Min-Ant-System*

Description

Construct subtests from a given pool of items using the classical Max-Min Ant-System (Stützle, 1998). Allows for multiple constructs, occasions, and groups.

Usage

```
mmas(data, factor.structure, capacity = NULL, item.weights = NULL,
  item.invariance = "congeneric", repeated.measures = NULL,
  long.invariance = "strict", mtmm = NULL, mtmm.invariance = "configural",
  grouping = NULL, group.invariance = "strict", auxiliary = NULL,
  use.order = FALSE, software = "lavaan", cores = NULL,
  objective = objective.preset, ignore.errors = FALSE, ants = 16,
  colonies = 256, evaporation = 0.95, alpha = 1, beta = 1,
  pheromones = NULL, heuristics = NULL, deposit = "ib",
  localization = "nodes", pbest = 0.005, tolerance = 0.5,
  schedule = "run", analysis.options = NULL, suppress.model = FALSE,
  seed = NULL, filename = NULL)
```

Arguments

data	A data.frame containing all relevant data.
factor.structure	A list linking factors to items. The names of the list elements correspond to the factor names. Each list element must contain a character-vector of item names that are indicators of this factor.
capacity	A list containing the number of items per subtest. This must be in the same order as the factor.structure provided. If a single number, it is applied to all subtests. If NULL all items are evenly distributed among the subtests.
item.weights	A placeholder. Currently all weights are assumed to be one.
item.invariance	A character vector of length 1 or the same length as factor.structure containing the desired invariance levels between items pertaining to the same subtest. Currently there are five options: 'congeneric', 'ess.equivalent', 'ess.parallel', 'equivalent', and 'parallel', the first being the default.
repeated.measures	A list linking factors that are repeated measures of each other. Repeated factors must be in one element of the list - other sets of factors in other elements of the list. When this is NULL (the default) a cross-sectional model is estimated.
long.invariance	A character vector of length 1 or the same length as repeated.measures containing the longitudinal invariance level of repeated items. Currently there are four options: 'configural', 'weak', 'strong', and 'strict'. Defaults to 'strict'. When repeated.measures=NULL this argument is ignored.

`mtmm` A list linking factors that are measurements of the same construct with different methods. Measurements of the same construct must be in one element of the list - other sets of methods in other elements of the list. When this is `NULL` (the default) a single method model is estimated.

`mtmm.invariance`
A character vector of length 1 or the same length as `mtmm` containing the invariance level of MTMM items. Currently there are five options: 'none', 'configural', 'weak', 'strong', and 'strict'. Defaults to 'configural'. With 'none' differing items are allowed for different methods. When `mtmm=NULL` this argument is ignored.

`grouping` The name of the grouping variable. The grouping variable must be part of `data` provided and must be a numeric variable.

`group.invariance`
A single value describing the assumed invariance of items across groups. Currently there are four options: 'configural', 'weak', 'strong', and 'strict'. Defaults to 'strict'. When `grouping=NULL` this argument is ignored.

`auxiliary` The names of auxiliary variables in `data`. These can be used in additional modeling steps that may be provided in `analysis.options$model`.

`use.order` A logical indicating whether or not to take the selection order of the items into account. Defaults to `FALSE`.

`software` The name of the estimation software. Can currently be 'lavaan' (the default) or 'Mplus'. Each option requires the software to be installed.

`cores` The number of cores to be used in parallel processing. If `NULL` (the default) the result of `detectCores` will be used. On Unix-y machines parallel processing is implemented via `mclapply`, on Windows machines it is realized via `parLapply`.

`objective` A function that converts the results of model estimation into a pheromone. See 'details' for... details.

`ignore.errors` A logical indicating whether or not to ignore estimation problems (such as non positive-definite latent covariance matrices). Defaults to `FALSE`.

`ants` The number of ants per colony to be estimated. Can either be a single value or an array with two columns for parameter scheduling. See 'details'.

`colonies` The maximum number of colonies estimated since finding the latest global-best solution before aborting the process. Can either be a single value or an array with two columns for parameter scheduling. See 'details'.

`evaporation` The evaporation coefficient. Can either be a single value or an array with two columns for parameter scheduling. See 'details'.

`alpha` The nonlinearity coefficient of the pheromone-trail's contribution to determining selection probabilities. Defaults to 1 (linear). Can either be a single value or an array with two columns for parameter scheduling. See 'details'.

`beta` The nonlinearity coefficient of the heuristics' contribution to determining selection probabilities. Defaults to 1 (linear). Can either be a single value or an array with two columns for parameter scheduling. See 'details'.

`pheromones` A list of pheromones as created by `mmas`. This can be used to continue previous runs of this function.

`heuristics` An object of the class `stuartHeuristic` as provided by `heuristics` which contains heuristic information to be used in determining selection probabilities. If `NULL` (the default) selection probabilities are determined solely by the pheromones.

`deposit` Which deposit rule to use. Can be either 'ib' (the default) for an iteration-best deposit rule, or 'gb' for a global-best deposit rule.

`localization` Which localization to use when depositing pheromones. Can be either 'nodes' (the default) for depositing pheromones on selected nodes or 'arcs' for depositing on selection arcs.

`pbest` The desired overall probability of constructing the global-best solution when the algorithm convergels. Can either be a single value or an array with two columns for parameter scheduling. See 'details'.

`tolerance` The tolerance of imprecision when comparing the pheromones to the upper and lower limits. Can either be a single value or an array with two columns for parameter scheduling. See 'details'.

`schedule` The counter which the scheduling of parameters pertains to. Can be either 'run' (the default), for a continuous schedule, 'colony', for a schedule that is restarted every time a new global best is found, or 'mixed' for a schedule that restarts its current phase every time a new global best is found. See 'details'.

`analysis.options`
A list additional arguments to be passed to the estimation software. The names of list elements must correspond to the arguments changed in the respective estimation software. E.g. `analysis.options$model` can contain additional modeling commands - such as regressions on auxiliary variables.

`suppress.model` A logical indicating whether to suppress the default model generation. If `TRUE` a model must be provided in `analysis.options$model`.

`seed` A random seed for the generation of random samples. See `Random` for more details.

`filename` The stem of the filenames used to save inputs, outputs, and data files when `software='Mplus'`. This may include the file path. When `NULL` (the default) files will be saved to the temporary directory, which is deleted when the R session is ended.

Details

The pheromone function provided via `objective` is used to assess the quality of the solutions. These functions can contain any combination of the fit indices provided by the estimation software. When using Mplus these fit indices are 'rmsea', 'srmr', 'cfi', 'tli', 'chisq' (with 'df' and 'pvalue'), 'aic', 'bic', and 'abic'. With lavaan any fit index provided by `inspect` can be used. Additionally 'crel' provides an aggregate of composite reliabilites, 'rel' provides a vector or a list of reliability coefficients for the latent variables, 'con' provides an aggregate consistency estimate for MTMM analyses, and 'lvcor' provides a list of the latent variable correlation matrices. For more detailed objective functions 'lambda', 'theta', 'psi', and 'alpha' provide the model-implied matrices. Per default a pheromone function using 'crel', 'rmsea', and 'srmr' is used. Please be aware that the `objective` must be a function with the required fit indices as (correctly named) arguments.

The scheduling of parameters is possible for the arguments ants, colonies, evaporation, pbest, alpha, beta, tolerance, and deposit. For all of these parameter scheduling is done when an array with two columns is provided. The first column of the array contains the timer, i.e. when to switch between parameter settings, the second column contains the values. The argument schedule can be used to select an absolute schedule (schedule='run'), a relative schedule which resets completely after a new global best is found (schedule='colony'), or a mixed version which resets the current phase of the schedule after a new global best is found (schedule='mixed'). When providing a parameter schedule for iterations 0, 3, and 10 using 'run' will result in a change after the third and the tenth iteration - irrespective of whether global best solutions were found. In contrast, using 'colony' will result in the first setting being used again once a new global best is found. This setting will then be used until iteration 3 (if no new best solution is found) before a switch occurs. If a new global best is found the setting will begin the sequence from the beginning. Using 'mixed' will result in the first setting being used until three consecutive iterations cannot produce a new global best. After this the second setting is used. If a new global best is found, the second setting is kept, but for the purpose of the schedule it is now iteration 3 again, meaning that the third setting will be used later than in a 'run' schedule.

Value

Returns an object of the class stuartOutput for which specific summary and plot methods are available. The results are a list.

call	The called function.
software	The software used to fit the CFA models.
parameters	A list of the ACO parameters used.
analysis.options	A list of the additional arguments passed to the estimation software.
timer	An object of the class proc_time which contains the time used for the analysis.
log	A data.frame containing the optimization history.
solution	A list of matrices with the choices made in the global-best solution.
pheromones	A list of matrices with the pheromones of each choice.
subtests	A list containing the names of the selected items and their respective subtests.
final	The results of the estimation of the global-best solution.

Author(s)

Martin Schultze

References

Stützle, T. (1998). Local search algorithms for combinatorial problems: Analysis, improvements, and new applications. Unpublished doctoral dissertation. Darmstadt: Fachbereich Informatik, Universität Darmstadt.

See Also

bruteforce, gene, randomsamples, heuristics

Examples

```
# MMAS in a simple situation
# requires lavaan
# number of cores set to 1 in all examples
data(fairplayer)
fs <- list(si = names(fairplayer)[83:92])

# minimal example
sel <- mmas(fairplayer, fs, 4,
  colonies = 0, ants = 10,  # minimal runtime, remove for application
  seed = 55635, cores = 1)
summary(sel)


# longitudinal example
data(fairplayer)
fs <- list(si1 = names(fairplayer)[83:92],
  si2 = names(fairplayer)[93:102],
  si3 = names(fairplayer)[103:112])

repe <- list(si = c('si1', 'si2', 'si3'))

# change evaporation rate after 10 and 20 colonies
sel <- mmas(fairplayer, fs, 4,
  repeated.measures = repe, long.invariance = 'strong',
  evaporation = cbind(c(0, 10, 20), c(.95, .8, .5)),
  seed = 55635, cores = 1)

# adding a predictor variable to selection model (using lavaan)
data(fairplayer)
fs <- list(si = names(fairplayer)[83:92])

add <- 'si ~ IGS'

sel <- mmas(fairplayer, fs, 4,
  auxiliary = 'IGS',
  analysis.options = list(model = add),
  seed = 55635, cores = 1)

# inspect regression (in lavaan)
lavaan::coef(sel$final)

# same example, maximizing regression weight
obj <- function(chisq, df, pvalue, rmsea, srmr, beta) {
  beta[1, 'IGS']
}

sel <- mmas(fairplayer, fs, 4,
  auxiliary = 'IGS',
  analysis.options = list(model = add),
  objective = obj,
  seed = 55635, cores = 1)
```

```
# inspect regression (in lavaan)
lavaan::coef(sel$final)
```

`randomsamples` *Generating random samples of Subtests*

Description

Construct a defined number of random subtests from a given pool of items.

Usage

```
randomsamples(data, factor.structure, capacity = NULL,
  item.invariance = "congeneric", repeated.measures = NULL,
  long.invariance = "strict", mtmm = NULL, mtmm.invariance = "configural",
  grouping = NULL, group.invariance = "strict", auxiliary = NULL,
  use.order = FALSE, software = "lavaan", cores = NULL,
  objective = objective.preset, ignore.errors = FALSE,
  analysis.options = NULL, suppress.model = FALSE, seed = NULL,
  request.override = 10000, filename = NULL, n = 1000, percentile = 100)
```

Arguments

`data` A data.frame containing all relevant data.

`factor.structure`
A list linking factors to items. The names of the list elements correspond to the factor names. Each list element must contain a character-vector of item names that are indicators of this factor.

`capacity` A list containing the number of items per subtest. This must be in the same order as the `factor.structure` provided. If a single number, it is applied to all subtests. If `NULL` all items are evenly distributed among the subtests.

`item.invariance`
A character vector of length 1 or the same length as `factor.structure` containing the desired invariance levels between items pertaining to the same subtest. Currently there are five options: 'congeneric', 'ess.equivalent', 'ess.parallel', 'equivalent', and 'parallel', the first being the default.

`repeated.measures`
A list linking factors that are repeated measures of each other. Repeated factors must be in one element of the list - other sets of factors in other elements of the list. When this is `NULL` (the default) a cross-sectional model is estimated.

`long.invariance`
A character vector of length 1 or the same length as `repeated.measures` containing the longitudinal invariance level of repeated items. Currently there are four options: 'configural', 'weak', 'strong', and 'strict'. Defaults to 'strict'. When `repeated.measures=NULL` this argument is ignored.

`mtmm` A list linking factors that are measurements of the same construct with different methods. Measurements of the same construct must be in one element of the list - other sets of methods in other elements of the list. When this is `NULL` (the default) a single method model is estimated.

`mtmm.invariance`
A character vector of length 1 or the same length as `mtmm` containing the invariance level of MTMM items. Currently there are five options: 'none', 'configural', 'weak', 'strong', and 'strict'. Defaults to 'configural'. With 'none' differing items are allowed for different methods. When `mtmm=NULL` this argument is ignored.

`grouping` The name of the grouping variable. The grouping variable must be part of `data` provided and must be a numeric variable.

`group.invariance`
A single value describing the assumed invariance of items across groups. Currently there are four options: 'configural', 'weak', 'strong', and 'strict'. Defaults to 'strict'. When `grouping=NULL` this argument is ignored.

`auxiliary` The names of auxiliary variables in `data`. These can be used in additional modeling steps that may be provided in `analysis.options$model`.

`use.order` A logical indicating whether or not to take the selection order of the items into account. Defaults to `FALSE`.

`software` The name of the estimation software. Can currently be 'lavaan' (the default), 'Mplus', or 'Mplus Demo'. Each option requires the software to be installed.

`cores` The number of cores to be used in parallel processing. If `NULL` (the default) the result of `detectCores` will be used. On Unix-y machines parallel processing is implemented via `mclapply`, on Windows machines it is realized via `parLapply`.

`objective` A function that converts the results of model estimation into a pheromone. See `mmas` for details.

`ignore.errors` A logical indicating whether or not to ignore estimation problems (such as non positive-definite latent covariance matrices). Defaults to `FALSE`.

`analysis.options`
A list additional arguments to be passed to the estimation software. The names of list elements must correspond to the arguments changed in the respective estimation software. E.g. `analysis.options$model` can contain additional modeling commands - such as regressions on auxiliary variables.

`suppress.model` A logical indicating whether to suppress the default model generation. If `TRUE` a model must be provided in `analysis.options$model`.

`seed` A random seed for the generation of random samples. See `Random` for more details.

`request.override`
The maximum number of combinations for which the estimation is performed immediately, without an additional override request.

`filename` The stem of the filenames used to save inputs, outputs, and data files when `software='Mplus'`. This may include the file path. When `NULL` (the default) files will be saved to the temporary directory, which is deleted when the R session is ended.

n	The number of random samples to be drawn.
percentile	The percentile of the final solution reported among the viable solutions. Defaults to 100 (the best solution found).

Value

Returns an object of the class stuartOutput for which specific summary and plot methods are available. The results are a list.

call	The called function.
software	The software used to fit the CFA models.
parameters	A list of the parameters used.
analysis.options	A list of the additional arguments passed to the estimation software.
timer	An object of the class proc_time which contains the time used for the analysis.
log	A data.frame containing the estimation history.
solution	NULL
pheromones	NULL
subtests	A list containing the names of the selected items and their respective subtests.
final	The results of the estimation of the global-best solution.

Author(s)

Martin Schultze

See Also

bruteforce, mmas, gene

Examples

```
# Random samples in a simple situation
# requires lavaan
# number of cores set to 1 in all examples
data(fairplayer)
fs <- list(si = names(fairplayer)[83:92])

# 10 random solutions, report median solution
sel <- randomsamples(fairplayer, fs, 4,
  n = 10, percentile = 50,
  seed = 55635, cores = 1)
summary(sel)
```

sups *Data from a scale for Supervisor Support*

Description

A scale for supervisor support with 19 items. The scale consists of two subscales: carreer promotion (items 1 through 12) and feedback and goal setting (items 13 through 19).

Usage

```
sups
```

Format

A data frame with 411 observations on 20 variables. The first variable indicates the person ID, the following 19 all stem from the scale for Supervisor Support

Source

Janssen, A.B., Schultze, M., & Grötsch, A. (2015). Following the ants: Development of short scales for proactive personality and supervisor support by Ant Colony Optimization. European Journal of Psychological Assessment.

Index