Package ‘eatRest’

October 21, 2014

Type Package

Title eatRest

Version 0.0.10

Depends R (>= 2.14.0), eatTools, MASS, car, date, foreign, gdata,reshape2, sendmailR, xlsx, plyr, R.utils, parallel, psych,ggplot2

Date 2014-10-20

Author eat authors `<eat-commits@lists.r-forge.r-project.org>`

Maintainer eat authors `<eat-commits@lists.r-forge.r-project.org>`

Description remaining package formerly known as ``eat''

License GPL (>= 2)

LazyLoad yes

LazyData yes

OS_type windows

Repository R-Forge

Repository/R-Forge/Project eat

Repository/R-Forge/Revision 441

Repository/R-Forge/DateTimeStamp 2014-10-20 13:37:56

Date/Publication 2014-10-20 13:37:56

R topics documented:

`automateConquestModel` *automateConquestModel: Write all requirend Input for a single ConQuest Run.*

Description

`automateConquestModel` facilitates data analysis using the software ConQuest. It automatically writes ConQuest syntax, label, anchor and data files for a single model specified by several arguments in R. Moreover, a batch file is created to start the analysis. For automatically specifying and running several models in a row, see `automateModels`.

Usage

```
automateConquestModel(dat, ID, regression=NULL, DIF=NULL, group.var=NULL,
weight=NULL, items, na=list(items=NULL, DIF=NULL, HG=NULL, group=NULL,
weight=NULL), person.grouping=NULL, item.grouping=NULL, compute.fit = TRUE,
model.statement="item", m.model="1pl", Title = NULL, jobName, jobFolder,
subFolder=list(), dataName=NULL, anchor=NULL, pathConquest, method=NULL,
std.err=NULL ,distribution=NULL, n.plausible=NULL, set.constraints=NULL,
nodes=NULL, p.nodes=NULL, f.nodes=NULL, n.iterations=NULL, converge=NULL,
deviancechange=NULL, seed = NULL, name.unidim=NULL,  allowAllScoresEverywhere = FALSE,
equivalence.table="wle", use.letters=FALSE, checkLink=FALSE, verbose=TRUE,
export = list(logfile = TRUE, systemfile = TRUE, history = TRUE, covariance = TRUE,
reg_coefficients = TRUE, designmatrix = TRUE))
```

Arguments

`dat`	A data frame containing all variables necessary for analysis.
`ID`	Name or column number of the identifier (ID) variable.
`regression`	Names or column numbers of one or more context variables (e.g., sex, school). These variables will be used for latent regression in ConQuest.
`DIF`	Name or column number of one grouping variable for which differential item functioning analysis is to be done.
`group.var`	Names or column numbers of one or more grouping variables. Descriptive statistics for WLEs and Plausible Values will be computed separately for each group in ConQuest.
`weight`	Name or column number of one weighting variable.
`items`	Names or column numbers of variables with item responses.
`na`	A named list of numerical vectors indicating values to be considered as missing. Specific missing codes can be defined for each type of variable.
`item.grouping`	A named data frame indicating how items should be grouped to dimensions. The first column contains the names of all items and must be named `item`. The other columns contain dimension definitions and must be named with the respective dimension names. A value of 0 indicates that the respective item does not load on this dimension. A non-negative value indicates that the respective loads on this dimension with the specified weight. For examlpe, a value of 1.89 indicates that an item loads on this dimension with the weight 1.89.
`person.grouping`	A named data frame indicating which persons should be grouped. The first column contains the identifier variable and must have the same name as the respective column in `dat`. The other columns contain grouping definitions and must be named with the respective group names. A value of 1 indicates that a person belongs to this group, a value of 0 indicates that the respective person does not belong to this group.
`compute.fit`	Logical: Should fit statistics computed in ConQuest analysis?
`model.statement`	A character string with the model statement to use in the ConQuest syntax. If `model.statement == NULL`, the model statement in the ConQuest syntax is set to `item` by default. When a DIF variable is specified, the model statement is set to `item - [name of DIF variable] + item*[name of DIF variable]` by default. When the data format is polytomous (instead auf dichotomous), the model statement has to be formulated explicitly, for example `item + step` when a rating scale model is to be applied, or `item + item*step` when a partial credit model is to be applied. See ConQuest manual for details.
`m.model`	A character string specifying the IRT model used for analysis. At the time, only `"1PL"` is available.
`Title`	A character string with the analysis title for the ConQuest syntax. If `Title == NULL`, informations about computer and user name and R version are used as title.
`jobName`	A character string specifying the analysis name. All Conquest input and output files will named `jobName` with their corresponding extensions.

`jobFolder`	A character string specifying an already existing folder where all analysis files will be written to, for example `"C:/programme/analysis"`
`subFolder`	A named list of character strings specifying a maximum of two folders relative to `jobFolder` for data and output files. Character strings must be named `data` and `out`, for example `subFolder=list(data="../../dataset/analysis1", out="../../output/ar` If `subFolder$data == NULL`, the dataset is written to the folder specified by `jobFolder`. The same is true for `subFolder$out == NULL`.
`dataName`	A character string specifying the dataset name if it is intended to be different from the name specified by `jobName`. If `dataName == NULL`, the dataset is named `[jobName].dat`
`anchor`	A named data frame with anchor parameters. The first column contains the names of all anchor items and must be named `item`. The second column contains anchor parameters. Anchor items can be a subset of the items in the dataset and vice versa.
`pathConquest`	A character string with path and name of the ConQuest console, for example `"c:/programme/conquest/console_Feb2007.exe"` if `NULL` the newest executable in file.path(.Library,"eat/winexe/conquest") is used
`method`	A character string indicating which method should be used for analysis. Possible options are `"gauss"` (default), `"quadrature"` and `"montecarlo"`. See ConQuest manual pp.225 for details on these methods.
`std.err`	A character string specifying which type of standard error should be estimated. Possible options are `"full"`, `"quick"` (default) and `"none"`. See ConQuest manual pp.167 for details on standard error estimation.
`distribution`	A character string indicating the a priori trait distribution. Possible options are `"normal"` (default) and `"discrete"`. See ConQuest manual pp.167 for details on population distributions.
`n.plausible`	An integer value specifying the number of plausible values to draw. The default value is 5.
`set.constraints`	A character string specifying how the scale should be constrained. Possible options are `"cases"` (default), `"items"` and `"none"`. When anchor parameter are specified in `anchor`, constraints will be set to `"none"`.
`nodes`	An integer value specifying the number of nodes to be used in the analysis. The default value is 15.
`p.nodes`	An integer value specifying the number of nodes that are used in the approximation of the posterior distributions, which are used in the drawing of plausible values and in the calculation of EAP estimates. The default value is 2000.
`f.nodes`	An integer value specifying the number of nodes that are used in the approximation of the posterior distributions in the calculation of fit statistics. The default value is 2000.
`n.iterations`	An integer value specifying the maximum number of iterations for which estimation will proceed without improvement in the deviance. The minimum value permitted is 5. The default value is 20.
`converge`	An integer value specifiying the convergence criterion for parameter estimates. The estimation will terminate when the largest change in any parameter estimate between successive iterations of the EM algorithm is less than `converge`. The default value is 0.0001.

`deviancechange`	An integer value specifiying the convergence criterion for the deviance. The estimation will terminate when the change in the deviance between successive iterations of the EM algorithm is less than `deviancechange`. The default value is 0.0001.
`seed`	Sets the seed that is used in drawing random nodes for use in Monte Carlo estimation method. The default seed is 1.
`name.unidim`	A character string with the name of one latent dimension. Alternatively, the dimension name can be specified using the argument `item.grouping`.
`allowAllScoresEverywhere`	Logical: Relevant only in multidimensional models for polytomous data. If `FALSE`, different codes are allowed to occur in both dimensions, for example one dimension is measured with dichotomous items, i.e. 0/1, and the other dimension is measured with polytomous items, i.e. 1, 2, 3, 4. If `TRUE`, common codes for both dimension are used, i.e. 0, 1, 2, 3, 4 for both dimensions. In unidimensional models this argument has no effect.
`equivalence.table`	A character string specifying the type of equivalence table to print. Possible options are `"wle"` (default), `"mle"` and `NULL`.
`use.letters`	A logical value indicating whether item response values should be coded als letters. This option can be used in partial credit models comprising items with more than 10 categories to avoid response columns with width 2 in ConQuest.
`checkLink`	A logical value indicating whether the items in dataset are checked for being connected with each other via design. If `TRUE`, the function `checkLink` is called.
`verbose`	A logical value indicating whether messages are printed on the R console.
`export`	A named or unnamed list or vector of logical elements indicating whether Conquest should create logfile, systemfile, history file, covariance file, file of regression coefficients and file of designmatrix.

Details

If the folders specified in `subFolder` should be parent folders to `jobFolder`, they can be specified using double dots `..`. For example, if `jobFolder` is `"C:/programme/analysis"` and `subFolder` is `list(data="../dataset/analysis1", out="../../output/analysis1")`, dataset is written to `"C:/programme/dataset/analysis1"` and output is written to `"C:/output/analysis1"`.

Value

No results are returned to console. Input files and batch string are written to disk in specified folder(s).

Author(s)

Sebastian Weirich, Karoline Sachse, Martin Hecht

References

Wu, M.L., Adams, R.J., Wilson, M.R., & Haldane, S.A. (2007). *ACER ConQuest Version 2.0. Generalised Item Response Modeling Software*. Camberwell, Victoria: ACER Press.

See Also

`automateModels`, `checkLink`

Examples

```
## Not run:
#
# if software="conquest" (currently the only and default option) the path of the
#  windows executable ConQuest console must be specified by setting
# conquestParameters = list ("pathConquest"="<path_to_your_conquest.exe>")
# e.g. conquestParameters = list ("pathConquest"=""C:/ConQuest/console.exe"")
# if not explicitely specified it is searched for in
# file.path(.Library,"eat/winexe/conquest")
# e.g. "C:/R/R-2.14.2/library/eat/winexe/conquest"
# you can put your ConQuest executable there
#
# load example data
# (these are simulated achievement test data)
# Note: all examples corresponding to examples in help file of automateModels
data ( science1 )
#
### Example 1: create input files for a unidimensional Rasch model with all variables in dataset science1
# only variables of science1 you want to use for analysis must be classified. In unidimensional Rasch mode
# id and items have to be specified
# automateConquestModel needs data with collapsed missing
science1.collapsed <- collapseMissings(science1, items = science1.items)
dir.create("C:/temp")
ex1 <- automateConquestModel ( dat = science1.collapsed, ID = "id", items = science1.items,
jobFolder = "C:/temp", jobName = "rasch_unidim")
#
### Example 2: create input files for a multidimensional Rasch model with DIF
# option item.grouping specifies dimensions and mapping of items to dimensions
# item.grouping is a data.frame with item names in first column (item)
# and dimensions in further columns, mapping of items to dimension is
# indicated by 0 (item loads not on dimension) or 1 (item loads on dimension)
# (have a look at the example item.grouping science1.scales)
# since 6 dimensions are specified in science1.scales input for a 6-dimensional Rasch model is run
# running this example may take some time + convergence is suboptimal. This is only for illustration.
# Note: in higher dimensional modes, number of nodes increased to 2^[number of dimensions]. If not explicit
# specified by the user, automateModels automatically uses the estimator montecarlo, if nodes increased 35
# Note: As DIF variable(s) have to be numeric in Conquest, factor variables (e.g. "sex" with male/female)
# will be expressed as numeric indicator variables.
science1.collapsed <- collapseMissings(science1, items = science1.items)
dir.create("C:/temp")
ex2 <- automateConquestModel ( item.grouping = science1.scales, dat = science1.collapsed, ID = "id", items
 DIF = "sex", jobFolder = "C:/temp", jobName = "rasch_multidim" )

### Example 3: create input files for a multidimensional multigroup Rasch model with latent regression
# Note: As regression and group variables have to be numeric, factor variables (e.g. "sex" with male/femal
# will be expressed as numeric indicator variables.
# Moreover, unless estimation method is not specified explicitly, automateConquestModel chooses montecarlo
# estimation as gaussian quadrature is not available due to latent regression model and Bock-Aitken would
# 11390625 nodes. Note: As montecarlo needs to fix all item parameter when latent regression is applied, t
# parameters estimated in example 2 are used as anchor parameters here.
# Warning: This example may take a considerable amount of time. Its only for instruction.

science1.collapsed <- collapseMissings(science1, items = science1.items)
# in Conquest, latent regressors have to be numeric
dir.create("C:/temp")
# Run example 2 to gain item parameters
```

```
ex2 <- automateConquestModel ( item.grouping = science1.scales, dat = science1.collapsed, ID = "id", items
 DIF = "sex", jobFolder = "C:/temp", jobName = "rasch_multidim" )
setwd("C:/temp")
system ( "rasch_multidim.bat" , wait = TRUE , show.output.on.console = FALSE , invisible = FALSE )
### get ANCHOR parametern
prm <-  get.shw("rasch_multidim.shw")[[1]][,2:3]
ex3 <- automateConquestModel ( item.grouping = science1.scales, dat = science1.collapsed, ID = "id", items
 anchor = prm, group = "track", regression = "grade", jobFolder = "C:/temp", jobName = "rasch_multidim_regr
#
### Example 4: create input files for a multidimensional partial credit model with latent regression and un
# Warning: This example does not really make sense with regard to contents. Its only to illustrate generati
science1.collapsed <- collapseMissings(science1, items = science1.items)
# generate unequal factor loadings
science1.scales[c(10,14,20,22,1),2] <- 0.8
science1.scales[c(8,12,17), 2]      <- 1.27
science1.scales[c(25,29,33), 3]      <- 0.71
science1.scales[c(87,91,92), 5]      <- 0.97

# generate a polytomous structure
for (i in science1.scales[,"item"]) {science1.collapsed[!is.na(science1.collapsed[,i]),i] <- sample(c(0:3),

dir.create("C:/temp")
ex4 <- automateConquestModel ( item.grouping = science1.scales, dat = science1.collapsed, ID = "id", items
 model.statement = "item + item*step", method = "quadrature", regression = "sex", jobFolder = "C:/temp", jo

## End(Not run)
```

---

automateModels *automateModels*

Description

specify and run several ConQuest models

Usage

```
automateModels( dat, id = NULL, context.vars = NULL, items = NULL,
item.grouping = NULL, select.item.group = NULL, person.grouping.vars = NULL,
person.grouping.vars.include.all = FALSE, person.grouping = NULL,
select.person.group = NULL, checkLink = FALSE, additional.item.props = NULL,
folder, overwrite.folder = TRUE, analyse.name.prefix = NULL,
analyse.name = NULL, analyse.name.elements = NULL, data.name = NULL,
m.model = NULL, software = NULL, dif = NULL, weight = NULL, anchor = NULL,
regression = NULL, adjust.for.regression = TRUE, q3 = FALSE,
q3.p.est = c ( "wle" , "pv" , "eap" ), icc = FALSE, missing.rule = NULL,
cross = NULL, subfolder.order = NULL, subfolder.mode = NULL,
allNAdelete = TRUE, additionalSubFolder = NULL, run = TRUE, run.mode = NULL,
n.batches = NULL, run.timeout = 1440, run.status.refresh = 0.2,
cores = NULL, email = NULL, smtpServer = NULL, write.txt.dataset = FALSE,
write.xls.results = TRUE, delete.folder.countdown = 5,
conquestParameters = NULL )
```

Arguments

`dat`	data.frame containing all variables type of variables ("id" , "context.vars" or "items") must be set using options `id`, `context.vars`, `items`
`id`	name or column number of 'id' variable in `dat`
`context.vars`	names or column numbers of 'context' variables (e.g. sex, school , ...) in `dat`
`items`	names or column numbers of 'item' variables in `dat` if omitted, all variables that are not classified as 'id' or 'context' variables are treated as 'items'
`item.grouping`	data.frame with grouping information of items, first column must be 'item' which includes item names, further columns contain scale definitions, 0 indicates that the respective item is NOT part of the scale, 1 indicates that this item is part of the scale, colnames of columns are the names of the scales
`select.item.group`	character vector of scale names chosen for analysis
`person.grouping.vars`	character vector of 'context' variables in dataset which are used to automatically generate 'person.grouping', each category is transformed into the 'person.grouping' format
`person.grouping.vars.include.all`	logical vector (along person.grouping.vars), indicates whether to generate a variable 'all' for the specific variable
`person.grouping`	data.frame with grouping information of persons, first column must be the name of 'id' (e.g. idstud), further columns contain group definitions, 0 indicates that the respective person is NOT part of the group, 1 indicates that this person is part of the group, colnames of columns are the names of the groups
`select.person.group`	character vector of group names chosen for analysis
`checkLink`	logical: If `TRUE`, items in dataset are checked for being connected with each other via design (function `checkLink` is called) 23.02.2012: not yet implemented
`additional.item.props`	data.frame of additional item information to be merged to model results, first column must be 'item' and contain item names
`folder`	folder to write output into
`overwrite.folder`	logical, if TRUE (default), folder is completely emptied
`analyse.name.prefix`	prefix (e.g. "pilotStudy") to be attached to all analyses names
`analyse.name`	analyses names are usually automatically set, if you want to set them manually use this option
`analyse.name.elements`	analyses names are set automatically using these elements: c ("scale" , "group" , "dif" , "regression" , "anchor"), use this option to change composition and order of the analyses names generation
`data.name`	optional: character string specifying name of dataset if intend to differ from name specified by jobName. When dataName == NULL, dataset is named [jobName].dat
`m.model`	measurement model, "1pl" (default), "2pl", "3pl", "4pl"

`software`	"conquest" (default) no other software implemented yet
`dif`	variable that is used for differential item functioning
`weight`	case weight variable
`anchor`	data.frame with anchor information
`regression`	variable(s) that is/are used
`adjust.for.regression`	center plausible values and items on grand mean
`q3`	Logical: If `TRUE`, Yen's Q3 statistic is computed.
`q3.p.est`	person estimates that are used in q3 calculation, default: wle
`icc`	Logical: If `TRUE`, pdfs of item icc are generated.
`missing.rule`	definition how to recode distinct missings in dataset
`cross`	scales in 'item.grouping' and groups in 'person.grouping' can be crossed to define distinct analyses "all": scales and groups are crossed "item.groups", scales are separately (unidimensional) run (instead of one multidimensional model) "person.groups", person groups are separately (single group) run (instead of one multigroup model)
`subfolder.order`	subfolders are automatically generated in this order c ("i.model" , "p.model" , "m.model" , "software" , "dif" , "regression" , "anchor")
`subfolder.mode`	"none": no subfolders are created "full": complete subfolders are created according to 'subfolder.order' "intelligent" (default): meaningful subfolders are created
`allNAdelete`	if `TRUE` all cases with complete missings on `items` are removed, if `FALSE` these cases are not deleted Note: this is a global option, that is set for all modelss
`additionalSubFolder`	specification for 'data' and 'out' subfolder (constant over all analyses)
`run`	logical, if `TRUE` (default) models are run, if `FALSE` only syntax is created and batches are returned
`run.mode`	"serial": serial runs on local machine. see option 'cores' to specify number of parallel runs "parallel": batch files must be started manually (e.g. on several machines). see option 'n.batches' to specify number batch files
`n.batches`	if `run.mode`="parallel", number of batch files that are created, batch files contain one or more analyses
`run.timeout`	minutes to wait for analyses to finish, default: 1440 (24h)
`run.status.refresh`	time for console refresh of model run status, default: 0.2 (12sec)
`cores`	if `run.mode`="serial" and multiple analyses are run, number of cores to use. if `cores=NULL` (default) all cores are used if number of cores specified is greater than number of actual cores, number of actual cores is used
`email`	set email address to receive an email when analyses are finished or time's up
`smtpServer`	smtpServer for sending emails, default: "mailhost.cms.hu-berlin.de"
`write.txt.dataset`	write out datasets as ascii, default: FALSE
`write.xls.results`	if `TRUE` (default) results are written to Excel files

`delete.folder.countdown`	countdown for deletion of 'folder', default: 5 (seconds)
`conquestParameters`	Set ConQuest parameters as a named list. Available option are: "compute.fit", "model.statement", "pathConquest", "method", "std.err", "distribution", "n.plausible", "set.constraints", "nodes", "p.nodes", "f.nodes", "n.iterations", "converge", "deviancechange", "equivalence.table", "use.letters", "checkLink", "export" See automateConquestModel documentation for details.

Details

To run several models list parameters as corresponding lists Explicitly list NULL if parameter should not be set or be defaulted See examples

Value

`run=TRUE`	returns results in specific format
`run=FALSE`	path(es) to batch file(s) are returned as character vector

Author(s)

Martin Hecht, Karoline Sachse, Sebastian Weirich, Christiane Penk, Malte Jansen, Sebastian Wurster

Examples

```
## Not run:
# folder must be specified, WARNING: this folder is deleted by automateModels!!!
#
# if software="conquest" (currently the only and default option) the path of the
#  windows executable ConQuest console must be specified by setting
# conquestParameters = list ("pathConquest"="<path_to_your_conquest.exe>")
# e.g. conquestParameters = list ("pathConquest"="C:/ConQuest/console.exe")
# if not explicitely specified it is searched for in
# file.path(.Library,"eat/winexe/conquest")
# e.g. "C:/R/R-2.14.2/library/eat/winexe/conquest"
# you can put your ConQuest executable there
#
# load example data
# (these are simulated achievement test data)
data ( science1 )
#
### Example 1: running a unidimensional Rasch model with all variables in dataset science1
# all variables in science1 must be classified as either id, context.vars or items
# items may be omitted, then it is defaulted to variables that are not id or context.vars
ex1 <- automateModels ( dat = science1, id = "id", context.vars = science1.context.vars,
 folder = "C:/temp/automateModels/Example1" )
#
# item and person parameters can be obtained using \link{get.item.par} and \link{get.person.par}
item.par <- get.item.par ( ex1 )
person.par <- get.person.par ( ex1 )
#
### Example 2: running a multidimensional Rasch model
# option item.grouping specifies dimensions and mapping of items to dimensions
# item.grouping is a data.frame with item names in first column (item)
# and dimensions in further columns, mapping of items to dimension is
# indicated by 0 (item loads not on dimension) or 1 (item loads on dimension)
# (have a look at the example item.grouping science1.scales)
# since 6 dimensions are specified in science1.scales a 6-dimensional Rasch model is run
# this example runs some time + convergence is suboptimal
ex2 <- automateModels ( item.grouping = science1.scales, dat = science1, id = "id",
 context.vars = science1.context.vars, folder = "C:/temp/automateModels/Example2" )
#
### Example 3: running several unidimensional Rasch models in a row
# we use item.grouping = science1.scales with 6 dimensions
# instead of running one 6-dimensional model we will run 6 unidimensional models
# by specifying cross = "item.groups"
ex3 <- automateModels ( cross = "item.groups", item.grouping = science1.scales, dat = science1,
 id = "id", context.vars = science1.context.vars,
 folder = "C:/temp/automateModels/Example3" )
#
### Example 4: running 15 2-dimensional models (every scale combined with every other)
# Option select.item.group is used to specify various combinations of dimensions
# it is a list of 15 character vectors that incorporate scale names (from item.grouping data)
ex4 <- automateModels ( select.item.group =
 list ( c("BioKno","BioPro"),c("BioKno","CheKno"),c("BioKno","ChePro"),
 c("BioKno","PhyKno"),c("BioKno","PhyPro"),c("BioPro","CheKno"),c("BioPro","ChePro"),
 c("BioPro","PhyKno"),c("BioPro","PhyPro"),c("CheKno","ChePro"),c("CheKno","PhyKno"),
 c("CheKno","PhyPro"),c("ChePro","PhyKno"),c("ChePro","PhyPro"),c("PhyKno","PhyPro") ),
 item.grouping = science1.scales, dat = science1,
 id = "id", context.vars = science1.context.vars,
 folder = "C:/temp/automateModels/Example4" )
#
### Example 5: running Rasch models for several person subgroups
# we specify person.grouping.vars = "grade" to run seperate analysis for every value of grade (9/10)
# to include the complete analysis (all grades) person.grouping.vars.include.all is set to TRUE
# to trigger separate person subgroup analyses cross must be set to "person.groups"
# with this specification 3 models are run: all grades (9 and 10), grade 9, grade 10
ex5 <- automateModels ( person.grouping.vars = "grade",
 person.grouping.vars.include.all = TRUE,
 cross = "person.groups",
 dat = science1, id = "id", context.vars = science1.context.vars,
 folder = "C:/temp/automateModels/Example5" )
#
### Example 6: running Rasch models for several person subgroups and scales
# cross = "all" triggers unidimensional models with the combination of scales and person subgroups
# in this example every scale is run with grade 9 and with grade 10 separately (=12 models)
ex6 <- automateModels ( person.grouping.vars = "grade",
 item.grouping = science1.scales,
 cross = "all",
 dat = science1, id = "id", context.vars = science1.context.vars,
 folder = "C:/temp/automateModels/Example6" )


## End(Not run)
```

`bi.linking` *bi.linking*

Description

Links results from several analysis. Each analysis is linked with each other.

Usage

```
bi.linking ( results , folder=NULL , file.name=NULL , method = NULL , lower.triangle = TRUE , sc
```

Arguments

`results`	result list from automateModels run
`folder`	output folder, will be emptied!
`file.name`	file.name for output excel, default: "bi.linking.results.xlsx"
`method`	set linking method to either "Mean-Mean" , "Haebara" or "Stocking-Lord" (default)
`lower.triangle`	set reference groups for the linking
`scales`	Character vector of scales for which linking should separately done. If NULL, all analysis in the results list are linked. Note: due to suboptimalities in development process, analysis name must contain 'scale'! use this option with care!!

Value

writes linking results to excel file. returns linking results as list.

Author(s)

Martin Hecht

Examples

```
## Not run:
# folder must be specified, WARNING: this folder is deleted by automateModels!!!
#
# load example data
# this is the results structure returned from running Example 5 of \link{automateModels}
# see there for details of analyses
#
data ( ex5 )
#
# ex5 contains the results of 3 analyses:
names(ex5)
# [1] "all.i__grade.10"  "all.i__grade.9"   "all.i__grade.all"
#
# each pair of these 3 analyses are linked together by bi.linking
# if not run together, you can easily combine analyses from seperate \link{automateModels} runs by calling
# in this case make sure that analysis names are unique
#
# start linking, results are written to folder and are returned
ex5_linked <- bi.linking ( ex5 , folder = "C:/temp/automateModels/Example5/Linking" , file.name = "ex5_link
#
#
# if you want to link analyses for which no \code{automateModels} results structure is available
# you can use \link{make.link.dummy} to create a structure similar to \code{automateModels} results structu
# this can be used as input for \code{bi.linking}
#
```

```
# e.g. lets add an additional analysis for which only item difficulty and standard errors are available
# this information must be in a data.frame, see \link{make.link.dummy} for details
dfr <- data.frame (
"item" = c ( "BioKno01" , "CheKno02" , "PhyKno03" ) ,
"b"    = c ( -3.14      , -2.24      , -3.42      ) ,
"b.se" = c ( 0.612      , 0.453      , 0.783      )
)
#
# create a results object
add <- make.link.dummy ( dfr , "additional" )
#
# add this object to Example 5
ex5add <- c ( add , ex5 )
#
# start the linking procedure with the additional analysis
ex5add_linked <- bi.linking ( ex5add , folder = "C:/temp/automateModels/Example5/Linking2" , file.name = "e

## End(Not run)
```

---

checkLink *checkLink*

Description

Checks whether items in a dataset are linked via design. This may be useful in multiple matrix sampling designs.

Usage

```
checkLink ( dat, remove.non.responser = FALSE, na = NA, verbose = TRUE)
```

Arguments

`dat`	A data.frame where all columns denote test items
`remove.non.responser`	logical: Should cases with missings on all items be deleted?
`na`	character string specifying values to be treat as missing by design
`verbose`	logical: Should output printed to console?

Value

A logical value, i.e. `TRUE` or `FALSE`, indicating whether items in dataset are linked to each other.

Author(s)

Sebastian Weirich

Examples

```
dat    <- data.frame(item01 = c(1,0,NA,NA,NA), item02 = c(NA,NA,1,0,0), item03 = c(1,NA,NA,NA,NA) )
result <- checkLink(dat)
dat    <- data.frame(dat, item04 = c(0, NA,NA,NA,1) )
result <- checkLink(dat)

# checkLink for each dimension in science1
data(science1)
results <- by (data = science1.item.characteristics, INDICES = science1.item.characteristics$scale, FUN = f
               collapsed <- collapseMissings(science1[,scales$item])
               results   <- checkLink(collapsed)
           })

# However, if only persons which anwered at least one item are considered
results <- by (data = science1.item.characteristics, INDICES = science1.item.characteristics$scale, FUN = f
               collapsed <- collapseMissings(science1[,scales$item])
               results   <- checkLink(collapsed, remove.non.responser = TRUE)
           })
```

---

compareModels *compare ConQuest models*

Description

retrieves model information (sample size, deviance, number of parameters) and calculates AIC and BIC; if more than 1 model is specified models are compared (AIC difference, BIC difference, Chi square prob.)

Usage

```
compareModels ( path , xlsx = NULL )
```

Arguments

path	either 1 of 4 inputs: [1] full path (directory + file name) of a ConQuest shw file [2] a list of ConQuest shw files (full paths) [3] a character vector of ConQuest shw files (full paths) [4] a folder in which it will be searched for ConQuest shw files
xlsx	full path (directory + file name) to Excel to be written (don't forget ".xlsx" suffix)

Value

returns a list of 2 data.frames: the first called 'models' contains model information; the second called 'model.comparison' contains information of model comparison

Author(s)

Martin Hecht

Examples

```
## Not run:
# just run any example of \link{automateModels}
# then:
#      compareModels ( <folder> )
# e.g. compareModels ( "C:/temp/automateModels/Example3" )

## End(Not run)
```

`ConQuest.Log.Example1.log.bz2`
Example Log File from ConQuest

Description

This is a text file with the log from a ConQuest analysis It can be accessed via bzfile (file.path(.Library , "eat/extdata/ConQuest.Log.Example1.log.bz2"))

Format

txt

`detect.suppression` *detect suppression effects in regression models*

Description

This function detects suppression effects in regression models.

Usage

```
detect.suppression ( dat , dependent , independent , full.return = FALSE , xlsx.path = NULL )
```

Arguments

`dat`	data.frame with data to be used
`dependent`	dependent variable in regression model
`independent`	character vector of independent variables in regression model
`full.return`	if `FALSE` a data.frame as a quadratic matrix with suppression effects (TRUE/FALSE) of independent variables is returned if `TRUE` a data.frame with all calculated terms ist returned
`xlsx.path`	full path of Excel file that results should be written to

Details

formulae (13.39a) and (13.39b) decribed in Bortz (1999) page 446 are used

if `full.return=TRUE` a data.frame is returned.

Columns are:

rownames: <dependent variable> ~ <independent variables> | <independent variable that is tested for suppression>

multiple.reg: logical, indicates wether there are 2 (FALSE) or more than 2 (TRUE) independent variables in the regression model

dep: dependent variabel in regression model

pred: independent variable that is investigated on suppression effect

preds: independent variables in regression model besides `pred`

cor_pred_c: correlation of `pred` and dependent variable

cor_pred_fitted_c: correlation of predicted `pred` by indepenent variables and dependent variable

r.sq_pred: R squared from model predicting `pred` by independent variables

rterm.minus: right term in formula (13.39a)

rterm.plus: right termn in formula (13.39b)

rterm.minus.diff: difference of `rterm.minus` and `cor_pred_c`

rterm.plus.diff: difference of `cor_pred_c` and `rterm.plus`

(positive difference of `rterm.minus.diff` or `rterm.plus.diff` indicates suppression effect)

rterm.minus.log: logical value of formula (13.39a)

rterm.plus.log: logical value of formula (13.39b)

suppression: logical, `rterm.minus.log` | `rterm.plus.log`

if `full.return=FALSE` a data.frame as quadratic matrix is returned:

rows and columns are independent variables

diagonal includes `suppression` for suppression effect of variable in multiple regression

triangles include `suppression` for bivariate independent variables, "row" suppresses "column"

Value

depends on options `full.return`

Author(s)

Martin Hecht

References

for formulae used by `detect.suppression` see

Bortz, J. (1999). Statistik fuer Sozialwissenschaftler. 5. Auflage. Berlin: Springer. p. 446

dichotomize *dichotomize a numeric vector*

Description

dichotomize a numeric vector by median or mean split

Usage

```
dichotomize ( numvec , method = c("median","mean") , randomize = TRUE , ... )
```

Arguments

`numvec`	numeric vector
`method`	either median or mean split
`randomize`	logical, if `TRUE` elements that equal the split threshold are randomly assigned to one of the two groups if `FALSE` default behavior of cut is used
`...`	arguments are passed to set.seed and cut

Value

returns vector with dichotomization indicators

Author(s)

Martin Hecht

Examples

```
numvec <- c(1,2,3,4,5)
dichotomize ( numvec )

# set seed for random assignment of elements that match split threshold by passing argument seed to functi
# ( 3 in numvec is on threshold if median is used )
dichotomize ( numvec , seed = 12345 )

# set level names by passing argument labels to cut function
dichotomize ( numvec , labels = c ( "low" , "high") )
```

equating.rasch *Align Item Parameters from Separate Analyses*

Description

This function can be used to align two sets of item parameters from two different Rasch analyses (e.g., two populations of examinees of differing abilities) so that they are on the same scale. The item parameters of one group are transformed to the scale of the other group by adding a constant.

Usage

```
equating.rasch(x, y, theta = seq( -4, 4, len=100), method = c("Mean-Mean", "Haebara", "Stocking-
```

Arguments

`x`	A data.frame with item names and parameters for group 1. This is the group which will be linked to the scale of group 2. The data.frame has to follow the structure: First column contains item names for the focal group, second column contains item parameters, and, if `compute.dif = TRUE`, third colunm contains the standard errors of the item parameters.
`y`	A data.frame with item names and parameters for group 2. This is the group for which the scale is defined. The data.frame has to follow the structure: First column contains item names for the focal group, second column contains item parameters, and, if `compute.dif = TRUE`, third colunm contains the standard errors of the item parameters.
`theta`	theta values where the test characteristic curves are evaluated
`method`	Method for determining the linking constant, either `Mean-Mean`, `Haebara` or codeStocking-Lord
`compute.dif`	Logical: Whether differential item functioning in the two groups should be examined.

Details

`equating.rasch` provides three methods to determine this constant: `Mean-Mean` the difference of the item parameter means of both samples are obtained. `Haebara` additionally takes the difference between item characteristic curves into account. `Stocking-Lord` additionally takes the test characteristic functions in account, thus minimizing differences in expected scores rather than observed scores or parameters. In most practical applications, the three linking constants should be fairly similar.

When `compute.dif = TRUE`, differential item functioning (DIF) in anchor items is examined. This can be useful to examine items with large shifts, which can be subsequently excluded from the linking procedure. DIF is computed according to the formula in Lord (1980). Additionally, the magnitude of DIF is categorized as small, moderate or large according to criteria established by the Educational Testing Service (ETS): category A (small DIF) if |DIF| < 0.43 or not significantly > 0, category B (moderate DIF) if 0.43 < |DIF| < 0.64 and |DIF| significantly > 0, and category C (large DIF) if |DIF| > 0.64 and significantly > 0.43.

Value

A list with the following components:

`B.est`	Linking constants determined by all three methods
`descriptives`	A list with the number of items used for linking, linking variance and standard deviation, and the linking error
`anchor`	A data.frame with all item parameters used for linking from both samples and the transformed parameters for group 1. If `compute.dif = TRUE`, additional statistics for DIF are also included.
`transf.par`	A data.frame with all item parameters from both samples and the transformed parameters for group 1.

Author(s)

Alexander Robitzsch

References

Kolen, M. J. & Brennan, R. L. (2004). *Test equating, scaling, and linking: Methods and practices*. New York: Springer. Yen, W. M., & Fitzpatrick, A. R. (2006). *Item response theory*. In R. L. Brennan (Ed.), Educational Measurement (4th ed., pp. 111-153). Westport, CT: American Council on Education.

See Also

`bi.linking`

`ex5` *Example 5 results*

Description

R object created by running Example 5 of automateModels

Usage

```
data(ex5)
```

Format

complex list structure

`exploreDesign` *explore data design*

Description

explore data structure with respect to specific missing code (e.g. "missing by design")

Usage

```
exploreDesign ( dat , na = NA , id = NULL , itemsPerPerson = TRUE , personsPerItem = TRUE )
```

Arguments

`dat`	data.frame
`na`	missing specification
`id`	id variable in `dat` if exists
`itemsPerPerson`	logical , if `TRUE` items per person list is returned
`personsPerItem`	logical , if `TRUE` persons per item list is returned

Value

depends on `itemsPerPerson` and `personsPerItem` , if both are `TRUE` a list with both elements is returned

Author(s)

Martin Hecht

Examples

```
data(science1)
d <- science1[,!colnames(science1) %in% science1.context.vars]
design <- exploreDesign ( dat = d , na = "mbd" , id = "id" )
str(design)
```

`get.dsc` *Read ConQuest 'descriptives' Output Files.*

Description

Reads ConQuest files with descriptive statistics for the estimated latent variables generated by the 'descriptives' statement.

Usage

```
get.dsc(file)
```

Arguments

`file`	Character string with the name of the ConQuest descriptives file.

Value

A named list of n elements with n being the number of groups for which descriptive statistics were computed. The names of the list are the group names. Each list contains the following elements:

`single.values`	A data frame containing the group name, dimension names, the number of observations, mean, standard deviation and variance for each of the latent dimensions. If the file contains descriptive statistics for plausible values, the number of rows in the data frame corresponds to the number of plausible values.
`aggregates`	A data frame containing the group name, dimension names and aggregated statistics for the mean, standard deviation and variance for each of the latent dimensions as well as (in a separate row) their standard errors.

Author(s)

Sebastian Weirich

References

See pp.162 of Wu, M.L., Adams, R.J., Wilson, M.R., & Haldane, S.A. (2007). *ACER ConQuest Version 2.0. Generalised Item Response Modeling Software*. Camberwell, Victoria: ACER Press.

get.equ — *Reads equivalence table created in Conquest analysis.*

Description

Reads Conquest files comprising equivalence tables for MLE or WLE parameters.

Usage

```
get.equ(file)
```

Arguments

`file`	Character string of the Conquest equ-file.

Value

A list of n+1 elements, with n the number of dimensions in the analysis. Each element is a data.frame, whose name correponds to the name of the dimension the values belongs to. All data.frames except the last one give the transformation of each possible raw score to the WLE or MLE score including it's standard error. First column in each data.frame contains the raw score, second column the transformed WLE or MLE score, third columns it's standard error.

The last element of the list give some sparse information about the model specifications.

References

See Conquest manual, pp.162.

get.history — *Reads Conquest history files.*

Description

Reads Conquest history file comprising parameter estimates of each iteration.

Usage

```
get.history(file, shw.object)
```

Arguments

`file`	Character string of the Conquest history file.
`shw.object`	Optional: R-Object created by `get.shw()`. Necessary to label the columns of the history file.

Value

A data.frame according to the corresponding Conquest history file. First column comprises the iteration number, second column the deviance of the corresponding iteration. Estimates of model parameters are listed in further columns.

Author(s)

Sebastian Weirich

---

get.item.par *get item parameters*

Description

obtain item parameters from automateModels results

Usage

```
get.item.par ( results )
```

Arguments

`results`	return object from `automateModels`

Value

returns a data.frame with item parameters from `automateModels` run:

Author(s)

Martin Hecht

Examples

```
## Not run:
# example results structure from \code{automateModels}
data ( ex5 )
#
( item.par <- get.item.par ( ex5 ) )

## End(Not run)
```

---

get.itn *Read ConQuest 'itanal' Output Files*

Description

Reads ConQuest files comprising item analyses generated by the 'itanal' statement.

Usage

```
get.itn(file)
```

Arguments

`file`	Character string with the name of the ConQuest item analysis file.

Value

A data frame with one row per item response category containing the following columns:

`item.nr`	Number of the item in the analysis
`item.name`	Name of the item
`Label`	Response category label
`Score`	Score of this response category
`n.valid`	Total number of students who responded to this item
`Abs.Freq`	Number of students who gave this response
`Rel.Freq`	Number of students who gave this response as a percentage of the total number of respondents to the item
`p`	Percentage of students who answered this item correctly
`pt.bis`	Point-biserial for this response
`t.value`	T-Value of the significance test whether the point-biserial correlation is different from 0
`p.value`	p-Value of the significance test whether the point-biserial correlation is different from 0
`PV1.Avg.1`	Mean ability of students who gave this response (based on plausible values)
`PV1.SD.1`	Standard deviation of ability of students who gave this response (based on plausible values)
`pbc`	Item discrimination
`threshold`	Item threshold
`delta`	Item delta

If the model is multidimensional, the mean and standard deviation of the ability of students who gave the respective response will be shown for each dimension.

Author(s)

Sebastian Weirich

References

See pp.193 of Wu, M.L., Adams, R.J., Wilson, M.R., & Haldane, S.A. (2007). *ACER ConQuest Version 2.0. Generalised Item Response Modeling Software*. Camberwell, Victoria: ACER Press.

`get.latent.corr` *get latent correlations, covariance and variances*

Description

retrieve latent correlations, covariance and variances from ConQuest shw files

Usage

```
get.latent.corr ( path , xlsx = NULL , covariance = TRUE , variance = TRUE , sort = TRUE )
```

Arguments

`path`	either 1 of 4 inputs: [1] full path (directory + file name) of a ConQuest shw file [2] a list of ConQuest shw files (full paths) [3] a character vector of ConQuest shw files (full paths) [4] a folder in which it will be searched for ConQuest shw files
`xlsx`	full path (directory + file name) to Excel to be written (don't forget ".xlsx" suffix)
`covariance`	logical: should covariance(s) be extracted (default: TRUE)
`variance`	logical: should variance(s) be extracted (default: TRUE)
`sort`	logical: if `TRUE` (default) the latent correlation matrix ist sorted as in the analysis with most dimensions; if `FALSE` the latent correlation matrix ist not sorted, instead the order is determined by the order of dimensions in the analyses (first to last)

Value

returns a data.frame with correlations, covariance and variances of ConQuest analysis/analyses

Author(s)

Martin Hecht

Examples

```
## Not run:
# just run any example of \link{automateModels}
# then:
#      get.latent.corr ( <folder> )
# e.g. get.latent.corr ( "C:/temp/automateModels/Example1" )

## End(Not run)
```

`get.person.par` *get person parameters*

Description

obtain person parameters from automateModels results

Usage

```
get.person.par ( results )
```

Arguments

`results`	return object from `automateModels`

Value

returns a data.frame with person parameters from `automateModels` run:

Author(s)

Martin Hecht

Examples

```
## Not run:
# example results structure from \code{automateModels}
data ( ex5 )
#
( person.par <- get.person.par ( ex5 ) )

## End(Not run)
```

---

get.plausible — *Read ConQuest Plausible Values Output Files*

Description

This function reads ConQuest plausible value files and automatically identifies the number of cases, the number of plausible values and the number of dimensions.

Usage

```
get.plausible(file, verbose = FALSE)
```

Arguments

`file`	Character string with the name of the ConQuest plausible values file.
`verbose`	Logical: Should messages printed on console?

Value

A data frame with one row per person containing the following columns:

`case`	Case number
`ID`	Identifier for this case
`pv`	Plausible value. Columns are named `pv.[name of dimension]_[number of plausible value].` For example, pv.reading_6 refers to the 6th plausible value of reading dimension.
`eap`	Expected*a posteriori* ability estimate for this person. Columns are named `eap.[name of dimension`
`eap.se`	Standard error of the EAP estimate. Columns are named `eap.se.[name of dimension]`

Author(s)

Sebastian Weirich

References

See pp.230 of Wu, M.L., Adams, R.J., Wilson, M.R., & Haldane, S.A. (2007). *ACER ConQuest Version 2.0. Generalised Item Response Modeling Software*. Camberwell, Victoria: ACER Press.

`get.q3` *get.q3*

Description

get Q3 statistics

Usage

```
get.q3 ( results )
```

Arguments

`results`	results (structured list) from automateModels run

Value

list (analyses) of data.frames in matrix format containing Q3 statistics

Author(s)

Martin Hecht

`get.shw` *Read ConQuest showfiles*

Description

Function reads Conquest showfiles and transforms them into a R list of data frames.

Usage

```
get.shw(file, dif.term = NULL, split.dif = TRUE,
        abs.dif.bound = 0.64, sig.dif.bound = 0.43)
```

Arguments

`file`	Character string of the Conquest showfile to be read in.
`dif.term`	Optional: Character string. Name of the term considered to be DIF-term. Must match corresponding term in showfile.
`split.dif`	Logical: When TRUE, DIF-Parameter are only given for Reference group.
`abs.dif.bound`	When DIF-Parameter are evaluated, this specifies the critical value for absolute DIF.
`sig.dif.bound`	When DIF-Parameter are evaluated, this specifies the critical value for confidence interval DIF.

Details

Funktion searches for 'TERM'-statements in Conquest showfile and reads the tables associated with. If one statement is specified to contain DIF analyses, absolute DIF value is computed 2*[group-specific parameter]. Confidence intervalls for 90, 95 and 99 percent are computed via the standard error of specific parameters. If both criteria - absolute DIF exceeds `abs.dif.bound` and the confidence intervall does not include `sig.dif.bound`, item is considered to have DIF.

Value

A list of data frames, named by the 'TERM'-statements in Conquest showfile, plus an additional data frame named `regression` with regression coefficients when latent linear regression model was specified in Conquest analysis, plus an additional data frame named `cov.structure` with covariance and correlation matrix of latent dimensions. If uni-dimensional model is specified, the variance of the latent dimension is given instead. If one term was specified as DIF-statement, the corresponding data frame is augmented with additional columns for confidence intervals and indicators specifying significant DIF.

Each data frame corresponding to a 'TERM' statement contains following columns:

`item.nr`	Item number
`item`	Name of item
`ESTIMATE`	Estimated difficulty of item
`ERROR`	Standard error of estimated item difficulty
`outfit`	Item's 'Outfit'
`outfit.ci.lb`	Lower bound of the outfit confidence interval
`outfit.ci.ub`	Upper bound of the outfit confidence interval
`outfit.t`	T-value for outfit
`infit`	Items's 'Infit'
`infit.ci.lb`	Lower bound of the infit confidence interval
`infit.ci.ub`	Upper bound of the infit confidence interval
`infit.t`	T-value for infit
`abs.dif`	Only for DIF analysis. Absolute DIF, computed as 2*[group-specific parameter].
`ci.lb`	Lower bound confidence interval for specific significance level of 90, 95 or 99 percent.
`ci.ub`	Upper bound confidence interval for specific significance level of 90, 95 or 99 percent.
`sig`	Indicates whether the corresponding item matches both DIF criteria. See details.

When latent regression was specified, the last element of the returned list is a data frame with regression coefficients, corresponding to the number of dimensions and the number of regressors. Regressor names, regression coefficients and its standard errors are given for each dimension.

Rows represent the regressors, columns represent the latent dimension to which the regression is fitted.

Author(s)

Sebastian Weirich

`get.wle` *Read ConQuest WLE or MLE Output Files.*

Description

Read Conquest files comprising maximum likelihood estimates (MLE) or weighted likelihood estimates (WLE).

Usage

```
get.wle(file)
```

Arguments

`file`	Character string with the name of the ConQuest MLE or WLE file.

Value

A data frame with one row per person containing the following columns.

`case`	Case number
`ID`	Identifier for this case
`n.solved`	Number of items this person answered correctly
`n.total`	Number of total items presented to this person
`wle`	WLE or MLE estimate. The last number of the columns name indicates the dimension the WLE or MLE estimate belongs to.
`wle.se`	Standard error of WLE or MLE estimate. The last number of the columns name indicates the dimension the WLE or MLE estimate belongs to.

Author(s)

Sebastian Weirich

References

See pp.230 of Wu, M.L., Adams, R.J., Wilson, M.R., & Haldane, S.A. (2007). *ACER ConQuest Version 2.0. Generalised Item Response Modeling Software*. Camberwell, Victoria: ACER Press.

`getConquestVersion` *get version (build) of ConQuest*

Description

get version (build) of ConQuest

Usage

```
getConquestVersion ( path.conquest , path.temp, asDate = TRUE )
```

Arguments

`path.conquest`	full path to ConQuest executable console
`path.temp`	optional: writeable folder used for temporary files. If not specified, R working directory will be used. Without writing access, NULL is returned.
`asDate`	if `TRUE` an object of class 'date' is returned if `FALSE` a character string is returned

Value

depends on option 'asDate'

Author(s)

Martin Hecht

Examples

```
getConquestVersion ( "c:/ConQuest/console_Feb2007.exe" )
```

`isConverged` *check convergence of ConQuest models*

Description

checks if ConQuest models in a directory have converged or not

Usage

```
isConverged ( path , txt = FALSE )
```

Arguments

`path`	main path of ConQuest models, or a path to a ConQuest shw-file
`txt`	if `TRUE` a convergence summary is written to convergence_summary.txt in `path`, and a file (either "_CONVERGED_" or "_N_O_T_CONVERGED_") is written to each model directory if `FALSE` a data.frame of convergence information is returned

Details

if `path` is a directory, `isConverged` checks recursively in `path` for shw files; alternatively `path` can be a full path to a single shw-file. models that converged, but the solution is not the best solution (ConQuest: "At termination the solution was not the best attained solution"), are treated as not converged

Value

depends on `txt` if no shw-files are found `NULL` is returned

Author(s)

Martin Hecht

---

log2init *Convert ConQuest Log to ConQuest Init*

Description

Convert a ConQuest logfile to ConQuest covariance, regression and item init files

Usage

```
log2init ( log.path , out.path = NULL , iteration = c("highestLikelihood","last","first") , out.
```

Arguments

`log.path`	full path to or connection of ConQuest logfile
`out.path`	path of output files , if `NULL` folder of log.path is defaulted
`iteration`	either "highestLikelihood" (default), "last" or "first", or a number
`out.files.suffix`	suffix to be added to output file names

Details

ConQuest tends to not completely write out log if running and option 'update = yes' is used. To avoid warnings and malfunction manually delete the last potentially incomplete iteration from log-file.

Value

writes files to `out.path`

Author(s)

Martin Hecht

Examples

```
## Not run:
log2init ( bzfile ( file.path( .Library , "eat/extdata/ConQuest.Log.Example1.log.bz2" ) ) , "c:/temp" )

## End(Not run)
```

long2matrix *long2matrix*

Description

transforms long format data.frame into a matrix format data.frame

Usage

```
long2matrix ( dat , sort = TRUE , triangle = NULL ,
force.diagonal = FALSE , exclude.diagonal = FALSE ,
long2matrix = TRUE )
```

Arguments

`dat`	data.frame with columns "row" , "col" , "val"
`sort`	sort rows and columns of matrix
`triangle`	if not `NULL` a symmetric matrix will be constructed available options are "upper" , "lower" , "both"
`force.diagonal`	a diagonal is forced into matrix even if no diagonal elements are in `dat`
`exclude.diagonal`	the diagonal is excluded if possible
`long2matrix`	if `FALSE` `dat` is not transformed

Details

WARNING: This function seems to be buggy. Do not use it or use it with care.

Value

`long2matrix = TRUE`	data.frame in matrix format
`long2matrix = FALSE`	data.frame in long format

Author(s)

Martin Hecht

Examples

```
d1 <- data.frame (
"row" = c ( "v1" , "v2" , "v2" , "v3" , "v1" , "v3" ) ,
"col" = c ( "v1" , "v3" , "v2" , "v1" , "v2" , "v3" ) ,
"val" = c ( 1 , 5 , 4 , 3 , 2 , 6 ) , stringsAsFactors = FALSE )

# unsorted matrix
long2matrix  ( dat = d1 , sort = FALSE )
# sorted by default
long2matrix  ( dat = d1 )
# extract upper triangle of symmetric matrix
```

```
long2matrix  ( dat = d1 , triangle = "upper" )
# exclude diagonal elements
long2matrix  ( dat = d1 , triangle = "upper" , exclude.diagonal = TRUE )
# if full matrix ("both" triangles) is requested, the diagonal cannot be excluded, option is ignored
long2matrix  ( dat = d1 , triangle = "both" , exclude.diagonal = TRUE )

# no diagonal elements are specified
d2 <- data.frame (
"row" = c ( "v2" , "v1" , "v1" ) ,
"col" = c ( "v3" , "v3" , "v2" ) ,
"val" = c ( 5 , 3 , 2 ) , stringsAsFactors = FALSE )

long2matrix ( dat = d2 )
# diagonal is set (with NAs)
long2matrix ( dat = d2 , triangle = "upper" , force.diagonal = TRUE )
```

`long2symmatrix` *long2symmatrix*

Description

transforms long format data.frame into a symmetric matrix format data.frame

Usage

```
long2symmatrix ( dat , sort = FALSE , triangle = c ("both","lower","upper") ,
 include.diagonal = TRUE , full.symmetric = FALSE )
```

Arguments

`dat`	data.frame with columns "row" , "col" , "val"
`sort`	sort rows and columns of matrix, can be either logical (if TRUE variables are alphatecially sorted) or a character vector that indicates order
`triangle`	if "lower" or "upper" only this triangle is extracted (the other is set to NA), if "both" both triangles are extracted
`include.diagonal`	logical, should diagonal elements be included or not
`full.symmetric`	if `TRUE`

Details

WARNING: This function has not been thoroughly tested. if `sort = FALSE` and `triangle = "lower"` matrix is sorted by occurence on dat$row if `sort = FALSE` and `triangle = "upper"` matrix is sorted by occurence on dat$col `triangle = "both"` implies `sort = TRUE`

Value

data.frame in "matrix format"

Author(s)

Martin Hecht

Examples

```
long <- data.frame (
"row" = c ( "v1" , "v2" , "v2" , "v3" , "v1" , "v3" ) ,
"col" = c ( "v1" , "v3" , "v2" , "v1" , "v2" , "v3" ) ,
"val" = c ( 1 , 5 , 4 , 3 , 2 , 6 ) , stringsAsFactors = FALSE )

long2symmatrix ( long )

long2symmatrix ( long , triangle = "lower" )
long2symmatrix ( long , triangle = "upper" )
long2symmatrix ( long , triangle = "both" )

long2symmatrix ( long , triangle = "lower" , include.diagonal = FALSE)
long2symmatrix ( long , triangle = "upper" , include.diagonal = FALSE)
long2symmatrix ( long , triangle = "both" , include.diagonal = FALSE)

long2symmatrix ( long , sort = c("v2","v3") )
long2symmatrix ( long , sort = c("v2","v3") , triangle = "lower" , include.diagonal = FALSE )

long2symmatrix ( long , full.symmetric = TRUE )
long2symmatrix ( long , full.symmetric = TRUE , triangle = "lower" )
long2symmatrix ( long , full.symmetric = TRUE , triangle = "lower" , include.diagonal = FALSE )
```

`make.link.dummy` *make.link.dummy*

Description

create a structure as input for bi.linking

Usage

```
make.link.dummy ( dfr , analysis.name = "dummy.analysis" , scale.name = "dummy.scale", group.nam
```

Arguments

`dfr`	data.frame with items, item difficulty ("b") and standard error of b ("b.se"); colnames must be "item", "b", "b.se"
`analysis.name`	name of analysis
`scale.name`	name of scale (dimension)
`group.name`	name of (person) group

Value

returns a structure as if created by automateModels with data contained in `dfr`; this object can be used as input for bi.linking

Author(s)

Martin Hecht

Examples

```
## Not run:
dfr <- data.frame (
"item" = c ( "BioKno01" , "CheKno02" , "PhyKno03" ) ,
"b"    = c ( -3.14      , -2.24      , -3.42      ) ,
"b.se" = c ( 0.612      , 0.453      , 0.783      )
)
link.dummy <- make.link.dummy ( dfr )

## End(Not run)
```

---

plotDevianceChange *plot deviance change*

Description

extract or plot (on console or to pdf) deviance change from ConQuest logfile

Usage

```
plotDevianceChange ( path , plot = TRUE , pdf = FALSE , out.path = NULL , extreme.crit = 0.75 )
```

Arguments

`path`	full path to or connection of ConQuest logfile, or just a path (in which ConQuest logfiles are to be (recursively) searched for)
`plot`	if `TRUE` deviance change plot is created
`pdf`	if `TRUE` plot ist written to pdf
`out.path`	path for pdf output file
`extreme.crit`	numeric, threshold criterion to remove outliers, is multiplied with standard deviation of deviance change

Details

ConQuest tends to not completely write out log if running and option 'update = yes' is used. To avoid warnings and malfunction manually delete the last potentially incomplete iteration from logfile. Points below 0 are red; if model converged (see `link{isConverged}` for details), the last point is larger and green

Value

depends on `plot` and `pdf`; if both are `FALSE` the deviance change data is returned, this is a named vector with names = iteration number and values = deviance change from previous iteration; if more than one ConQuest logfile is processed a list of named vectors is returned

Author(s)

Martin Hecht

Examples

```
## Not run:
plotDevianceChange ( path = file.path( .Library , "eat/extdata/ConQuest.Log.Example1.log.bz2" ) )
plotDevianceChange ( path = file.path( .Library , "eat/extdata/ConQuest.Log.Example1.log.bz2" ) , plot = F/

## End(Not run)
```

`plotDistributions` *plot latent distribution*

Description

creates latent distribution plots of two distributions on the same metric (e.g. persons and items)

Usage

```
plotDistributions ( distr1 , distr2 , distr1.name = "Persons" , distr2.name = "Items" , pdf = NU
```

Arguments

`distr1`	numeric vector of person estimates
`distr2`	numeric vector of item estimates
`distr1.name`	name of distribution 1 (e.g. "Persons")
`distr2.name`	name of distribution 2 (e.g. "Items")
`pdf`	pdf output file
`title`	title for graph
`scale.unit`	name of units
`distr1.color`	color of distribution 1
`distr2.color`	color of distribution 2
`alpha`	controls transparency of graph, see geom_density

Author(s)

Martin Hecht

Examples

```
## Not run:
persons <- rnorm ( 5000 )
items <- rnorm ( 200 ) + 1

plotDistributions ( persons , items )

## End(Not run)
```

`read.txt` *read text files*

Description

reads (compressed) text files with specific method

Usage

```
read.txt ( path , read.function = c ( "readLines" , "read.table" , "read.csv" , "read.csv2" , "r
```

Arguments

`path`	either directory, file, list of directories, list of files, vector of directories or vector of files
`read.function`	function to read in txt files, bzfile, read.table, read.csv, read.csv2, read.delim, or read.delim2
`file.ext`	character, extension of files that are to be read, extensions of compressed files are ignored, that means that `file.ext` refers to extension of uncompressed file and extension of compressed file without compression extension
`simplify`	logical, if `TRUE` list is unlisted if of length 1, if `FALSE` always a list is returned
`...`	arguments passed to function used as `read.function`; if path is (list/vector of) directory/ies, arguments 'all.files', 'recursive', 'pattern' and 'ignore.case' can be passed to list.files that is used to get all compressed files from directory

Details

reads text files with specific read function (`read.function`); compressed files are automatically uncompressed depending on their extension, see zip2con for supported compression types; if `path` is a single file that does not exists, it is searched for a compressed file with this name, if found this is used

Value

returns list of read in files (also see `simplify`); names of list are file names (full path) without compression extension; if `path` is not processable `NULL` is returned, or (when multiple files are processed) it is not appended to return list

Author(s)

Martin Hecht

Examples

```
## Not run:
fl <- file.path( .Library , "eat/extdata/ConQuest.Log.Example1.log.bz2" )
lns <- read.txt ( fl )
lns <- read.txt ( bzfile ( fl ) )
lns <- read.txt ( list ( fl , fl ) )
lns <- read.txt ( c ( fl , fl ) )
str ( lns )
```

```
## End(Not run)
```

`sortDatByNames` *sort data.frame by colnames and/or rownames*

Description

specify new colnames and/or rownames order, data.frame is sorted in accordance

Usage

```
sortDatByNames ( dat , col.order = NULL , row.order = NULL , warn = TRUE )
```

Arguments

`dat`	data.frame
`col.order`	character vector of colnames in new order
`row.order`	character vector of rownames in new order
`warn`	logical, if `TRUE` warnings are printed on output window if `col.order`/`row.order` do not correspond to colnames/rownames resp.

Value

data.frame

Author(s)

Martin Hecht

Examples

```
dat <- data.frame ( matrix ( rnorm ( 100 ) , ncol = 10 ) )
colnames ( dat ) <- paste ( "X" , 10:1 , sep = "" )
rownames ( dat ) <- paste ( "X" , 11:2 , sep = "" )
dat

# sort data.frame by col.order and row.order
sortDatByNames ( dat , paste ( "X" , 1:10 , sep = "" ) , paste ( "X" , 2:11 , sep = "" ) )
```

`source.it.all` *source.it.all*

Description

sources *.R files of `folder`

Usage

```
source.it.all ( folder="p:/ZKD/development" , use.zkd.conv = TRUE , development = TRUE , develop
```

Arguments

`folder`	folder with *.R files
`development`	if `TRUE` development versions are sourced (if non-existent the latest stable is sourced or nothing is sourced, see option `development.only`\ if `FALSE` stable versions are sourced
`use.zkd.conv`	if `TRUE` R files in `folder` are checked to be consisten with specific ("zkd") versioning convention \ if `FALSE` all R files in `folder` are sourced
`development.only`	if `TRUE` only development versions are sourced \ if `FALSE` stable versions are included
`exclude`	character vector of R files that should not be sourced

Value

sources R files

Author(s)

Martin Hecht, Christiane Penk

`userSpecifiedList` *userSpecifiedList*

Description

When a function requires several arguments as a list, `userSpecifiedList` is designed to 'match' users arguments to default argument structure.

Usage

```
userSpecifiedList(l, l.default, el.default = NULL)
```

Arguments

`l`	A named or unnamed list or vector of elements.
`l.default`	The default list of arguments needed by a function.
`el.default`	if length of `l` is longer than 1, `el.default` can be set to an numeric value to select one default element

Value

A list of arguments where the user specified arguments are matched into default arguments. If `el.default` is set, one element is returned.

Author(s)

Sebastian Weirich

Examples

```
default.arguments <- list(logfile = TRUE, systemfile = TRUE, history = TRUE, covariance = TRUE, reg_coeffi
users.arguments   <- c(FALSE, FALSE)
userSpecifiedList ( l = users.arguments, l.default = default.arguments)

users.arguments   <- list(history = FALSE)
userSpecifiedList ( l = users.arguments, l.default = default.arguments)

users.arguments   <- FALSE
names(users.arguments) <- "covariance"
userSpecifiedList ( l = users.arguments, l.default = default.arguments)
```

`yen.q3` *yen.q3*

Description

Q3 statistics

Usage

```
yen.q3 ( dat , theta , b , progress = T )
```

Arguments

`dat`	bla
`theta`	bla
`b`	bla
`progress`	bla

`zip2con` *convert compressed file(s) to connection(s)*

Description

convert compressed file(s) to a list of connection(s)

Usage

```
zip2con ( path , ... )
```

Arguments

`path`	either directory, file, list of directories, list of files, vector of directories or vector of files
`...`	arguments passed to file handling function bzfile; if path is (list/vector of) directory/ies, arguments 'all.files', 'recursive', and 'ignore.case' can be passed to list.files that is used to get all compressed files from directory

Details

function converts file(s) to connections by calling the appropriate uncompress function depending on file extension; currently supported are files with extensions "bz2" that are processed by bzfile

Value

returns list of connection(s); names of list are file names (full path) without compression extension; if `path` is already a connection it is returned as named list if `path` is not processable `NULL` is returned, or (when multiple files are processed) it is not appended to return list

Author(s)

Martin Hecht

Examples

```
## Not run:
fl <- file.path( .Library , "eat/extdata/ConQuest.Log.Example1.log.bz2" )
zip2con ( fl )
zip2con ( bzfile ( fl ) )
zip2con ( list ( fl , fl ) )
zip2con ( c ( fl , fl ) )

## End(Not run)
```

Index