Package 'BigSyn'

February 27, 2020

Type Package

Title X

Version 1.0

Date 2020-02-14

Author D. Bonnery

Maintainer D. Bonnery <dbonnery@umd.edu>

Description Data

Remotes tidyverse/magrittr,
       DanielBonnery/StudyDataTools

Depends devtools,
       ggplot2,
       sqldf,
       lattice,
       printr,
       knitr,
       reshape2,
       data.table,
       rlist,
       haven,
       sas7bdat,
       partykit

License GPL (>= 2)

LazyLoad yes

LazyData true

RoxygenNote 7.0.2

R topics documented:

Index **51**

augmentmaxT_f   *Creates cell marginal max.*

Description

Creates cell marginal max.

Usage

```
augmentmaxT_f(.data, variables, verbose = getOption("verbose"))
```

Arguments

.data	a dataframe
variables	a vector of character strings

Details

Assume one runs the program

augmentmaxT_f(.dataBigSyn::STtableA1,variables=c("AA.present")). The program looks for all the "cell variables" corresponding to "AA.present", by using the function BigSyn::get_var

The results is this:

AA.present_La_La_Lrn1, AA.present_La_Lb_Lrn1, AA.present_La_Lc_Lrn1, ... AA.present_Lc_La_Lrn4, AA.present_Lc_Lb_Lrn4, AA.present_Lc_Ld_Lrn4

The programs computes the number of marginal variables with the function looks for BigSyn::get_cellXXmarginscount. Here it is 3

The program creates the following character matrix, named patterns:

"1" "La" ""

"1" "Lb" ""

"1" "Lc" ""

"2" "La_La" "La"

"2" "La_Ld" "La"

"2" "Lb_Lb" "Lb"

"2" "Lc_La" "Lc"

"2" "Lc_Lb" "Lc"

"2" "Lc_Ld" "Lc"
"2" "La_Lb" "La"
"2" "La_Lc" "La"
"2" "Lb_La" "Lb"
"2" "Lb_Lc" "Lb"
"2" "Lb_Ld" "Lb"
"2" "Lc_Lc" "Lc"
"3" "La_La_Lrn1" "La_La"
"3" "La_Ld_Lrn1" "La_Ld"
"3" "Lb_Lb_Lrn1" "Lb_Lb"
"3" "Lc_La_Lrn1" "Lc_La"
"3" "Lc_Lb_Lrn1" "Lc_Lb"
"3" "Lc_Ld_Lrn1" "Lc_Ld"
"3" "La_Lb_Lrn1" "La_Lb"
"3" "La_Lc_Lrn1" "La_Lc"
"3" "Lb_La_Lrn1" "Lb_La"
"3" "Lb_Lc_Lrn1" "Lb_Lc"
"3" "Lb_Ld_Lrn1" "Lb_Ld"
"3" "Lc_Lc_Lrn1" "Lc_Lc"
"3" "La_La_Lrn2" "La_La"
"3" "La_Ld_Lrn2" "La_Ld"
"3" "Lb_Lb_Lrn2" "Lb_Lb"
"3" "Lc_La_Lrn2" "Lc_La"
"3" "Lc_Lb_Lrn2" "Lc_Lb"
"3" "Lc_Ld_Lrn2" "Lc_Ld"
"3" "La_Lb_Lrn2" "La_Lb"
"3" "La_Lc_Lrn2" "La_Lc"
"3" "Lb_La_Lrn2" "Lb_La"
"3" "Lb_Lc_Lrn2" "Lb_Lc"
"3" "Lb_Ld_Lrn2" "Lb_Ld"
"3" "Lc_Lc_Lrn2" "Lc_Lc"
"3" "La_La_Lrn3" "La_La"
"3" "La_Ld_Lrn3" "La_Ld"
"3" "Lb_Lb_Lrn3" "Lb_Lb"
"3" "Lc_La_Lrn3" "Lc_La"
"3" "Lc_Lb_Lrn3" "Lc_Lb"
"3" "Lc_Ld_Lrn3" "Lc_Ld"

"3" "La_Lb_Lrn3" "La_Lb"

"3" "La_Lc_Lrn3" "La_Lc"

"3" "Lb_La_Lrn3" "Lb_La"

"3" "Lb_Lc_Lrn3" "Lb_Lc"

"3" "Lb_Ld_Lrn3" "Lb_Ld"

"3" "Lc_Lc_Lrn3" "Lc_Lc"

"3" "La_La_Lrn4" "La_La"

"3" "La_Ld_Lrn4" "La_Ld"

"3" "Lb_Lb_Lrn4" "Lb_Lb"

"3" "Lc_La_Lrn4" "Lc_La"

"3" "Lc_Lb_Lrn4" "Lc_Lb"

"3" "Lc_Ld_Lrn4" "Lc_Ld"

Then for all i in 3:1 (starting with the maximum depth) list the different aggregations to the upper level to perform. So for i=3, aggregating to the second level will be done by computing the variables : AA.cont1_La_La, AA.cont1_La_Ld, AA.cont1_Lb_Lb, AA.cont1_Lc_La, AA.cont1_Lc_Lb, AA.cont1_Lc_Ld, AA.cont1_La_Lb, AA.cont1_La_Lc, AA.cont1_Lb_La, AA.cont1_Lb_Lc, AA.cont1_Lb_Ld, AA.cont1_Lc_Lc

For example AA.cont1_La_La =rowSums(.data([,c("AA.cont1_La_La_Lrn1", "AA.cont1_La_La_Lrn2", "AA.cont1_La_La_Lrn3", "AA.cont1_La_La_Lrn4"),drop=FALSE])

For i=2 aggregating to the upper level will be done by computing the variables : AA.cont1_La, AA.cont1_Lb, AA.cont1_Lc AA.cont1_La =rowSums(.data([,c("AA.cont1_La_La", "AA.cont1_La_Ld", "AA.cont1_La_Lb", "AA.cont1_La_Lc"),drop=FALSE])

For i=1 aggregating to theupper level will be done by computng the variable AA.cont1_=rowSums(.data([,c("AA.cont1_La", "AA.cont1_Lb", "AA.cont1_Lc"),drop=FALSE])

The computation of the marginal totals is done, the second step is the computation of the marginal ratios.

It is done by looping on the rows of the patterns matrix

Line j of pattern is a length 3 character vector. let call patterns[j,2] x and patterns[j,3] y The programs replaces the variable names paste0("AA.cont1",x) by the ration of the variable paste0("AA.cont1",x) by the variable named paste0("AA.cont1",y).

For example for the line "3" "La_Ld_Lrn3" "La_Ld", the following replacement will be made: AA.cont1_La_Ld_Lrn3=AA.cont1_La_Ld_Lrn3/AA.cont1_La_Ld

The same is applied to all the elements of the input parameter variables.

Value

a dataframe.

Examples

```
.data=BigSyn::STtableA1
variable="AA.present"
variables=variable
ASTtableA1<-augmentmaxT_f(.data,variables)
```

```
ASTtableA1[1:5,c("AA.present_","AA.present_La","AA.present_La_Lb")]
xx<-ASTtableA1[sort(grep("present",names(ASTtableA1),value=TRUE))]
xx[xx==0]<-NA
StudyDataTools::ggplot_missing(xx)
```

---

augmentpctT_f   *Convert cell totals to marginal ratios and create overall total.*

Description

Convert cell totals to marginal ratios and create overall total.

Usage

```
augmentpctT_f(.data, variables, verbose = getOption("verbose"))
```

Arguments

.data   a dataframe

variables   a vector of character strings

Details

Assume one runs the program

augmentpctT_f(.dataBigSyn::STtableA1,variables=c("AA.cont1","AA.cont1")). The program looks for all the "cell variables" corresponding to "AA,cont1", by using the function BigSyn::get_var

The results is this:

AA.cont1_La_La_Lrn1, AA.cont1_La_Ld_Lrn1, AA.cont1_Lb_Lb_Lrn1, AA.cont1_Lc_La_Lrn1, AA.cont1_Lc_Lb_Lrn1, AA.cont1_Lc_Ld_Lrn1, AA.cont1_La_Lb_Lrn1, AA.cont1_La_Lc_Lrn1, AA.cont1_Lb_La_Lrn1, AA.cont1_Lb_Lc_Lrn1, AA.cont1_Lb_Ld_Lrn1, AA.cont1_Lc_Lc_Lrn1, AA.cont1_La_La_Lrn2, AA.cont1_La_Ld_Lrn2, AA.cont1_Lb_Lb_Lrn2, AA.cont1_Lc_La_Lrn2, AA.cont1_Lc_Lb_Lrn2, AA.cont1_Lc_Ld_Lrn2, AA.cont1_La_Lb_Lrn2, AA.cont1_La_Lc_Lrn2, AA.cont1_Lb_La_Lrn2, AA.cont1_Lb_Lc_Lrn2, AA.cont1_Lb_Ld_Lrn2, AA.cont1_Lc_Lc_Lrn2, AA.cont1_La_La_Lrn3, AA.cont1_La_Ld_Lrn3, AA.cont1_Lb_Lb_Lrn3, AA.cont1_Lc_La_Lrn3, AA.cont1_Lc_Lb_Lrn3, AA.cont1_Lc_Ld_Lrn3, AA.cont1_La_Lb_Lrn3, AA.cont1_La_Lc_Lrn3, AA.cont1_Lb_La_Lrn3, AA.cont1_Lb_Lc_Lrn3, AA.cont1_Lb_Ld_Lrn3, AA.cont1_Lc_Lc_Lrn3, AA.cont1_La_La_Lrn4, AA.cont1_La_Ld_Lrn4, AA.cont1_Lb_Lb_Lrn4, AA.cont1_Lc_La_Lrn4, AA.cont1_Lc_Lb_Lrn4, AA.cont1_Lc_Ld_Lrn4

The programs computes the number of marginal variables with the function looks for BigSyn::get_cellXXmarginscount. Here it is 3

The program creates the following character matrix, named patterns:

"1" "La" ""

"1" "Lb" ""

"1" "Lc" ""

```
"2" "La_La" "La"
"2" "La_Ld" "La"
"2" "Lb_Lb" "Lb"
"2" "Lc_La" "Lc"
"2" "Lc_Lb" "Lc"
"2" "Lc_Ld" "Lc"
"2" "La_Lb" "La"
"2" "La_Lc" "La"
"2" "Lb_La" "Lb"
"2" "Lb_Lc" "Lb"
"2" "Lb_Ld" "Lb"
"2" "Lc_Lc" "Lc"
"3" "La_La_Lrn1" "La_La"
"3" "La_Ld_Lrn1" "La_Ld"
"3" "Lb_Lb_Lrn1" "Lb_Lb"
"3" "Lc_La_Lrn1" "Lc_La"
"3" "Lc_Lb_Lrn1" "Lc_Lb"
"3" "Lc_Ld_Lrn1" "Lc_Ld"
"3" "La_Lb_Lrn1" "La_Lb"
"3" "La_Lc_Lrn1" "La_Lc"
"3" "Lb_La_Lrn1" "Lb_La"
"3" "Lb_Lc_Lrn1" "Lb_Lc"
"3" "Lb_Ld_Lrn1" "Lb_Ld"
"3" "Lc_Lc_Lrn1" "Lc_Lc"
"3" "La_La_Lrn2" "La_La"
"3" "La_Ld_Lrn2" "La_Ld"
"3" "Lb_Lb_Lrn2" "Lb_Lb"
"3" "Lc_La_Lrn2" "Lc_La"
"3" "Lc_Lb_Lrn2" "Lc_Lb"
"3" "Lc_Ld_Lrn2" "Lc_Ld"
"3" "La_Lb_Lrn2" "La_Lb"
"3" "La_Lc_Lrn2" "La_Lc"
"3" "Lb_La_Lrn2" "Lb_La"
"3" "Lb_Lc_Lrn2" "Lb_Lc"
"3" "Lb_Ld_Lrn2" "Lb_Ld"
"3" "Lc_Lc_Lrn2" "Lc_Lc"
"3" "La_La_Lrn3" "La_La"
```

"3" "La_Ld_Lrn3" "La_Ld"

"3" "Lb_Lb_Lrn3" "Lb_Lb"

"3" "Lc_La_Lrn3" "Lc_La"

"3" "Lc_Lb_Lrn3" "Lc_Lb"

"3" "Lc_Ld_Lrn3" "Lc_Ld"

"3" "La_Lb_Lrn3" "La_Lb"

"3" "La_Lc_Lrn3" "La_Lc"

"3" "Lb_La_Lrn3" "Lb_La"

"3" "Lb_Lc_Lrn3" "Lb_Lc"

"3" "Lb_Ld_Lrn3" "Lb_Ld"

"3" "Lc_Lc_Lrn3" "Lc_Lc"

"3" "La_La_Lrn4" "La_La"

"3" "La_Ld_Lrn4" "La_Ld"

"3" "Lb_Lb_Lrn4" "Lb_Lb"

"3" "Lc_La_Lrn4" "Lc_La"

"3" "Lc_Lb_Lrn4" "Lc_Lb"

"3" "Lc_Ld_Lrn4" "Lc_Ld"

Then for all i in 3:1 (starting with the maximum depth) list the different aggregations to the upper level to perform. So for i=3, aggregating to the second level will be done by computing the variables : AA.cont1_La_La, AA.cont1_La_Ld, AA.cont1_Lb_Lb, AA.cont1_Lc_La, AA.cont1_Lc_Lb, AA.cont1_Lc_Ld, AA.cont1_La_Lb, AA.cont1_La_Lc, AA.cont1_Lb_La, AA.cont1_Lb_Lc, AA.cont1_Lb_Ld, AA.cont1_Lc_Lc

For example AA.cont1_La_La =rowSums(.data([,c("AA.cont1_La_La_Lrn1", "AA.cont1_La_La_Lrn2", "AA.cont1_La_La_Lrn3", "AA.cont1_La_La_Lrn4"),drop=FALSE])

For i=2 aggregating to the upper level will be done by computing the variables : AA.cont1_La, AA.cont1_Lb, AA.cont1_Lc AA.cont1_La =rowSums(.data([,c("AA.cont1_La_La", "AA.cont1_La_Ld", "AA.cont1_La_Lb", "AA.cont1_La_Lc"),drop=FALSE])

For i=1 aggregating to theupper level will be done by computng the variable AA.cont1_=rowSums(.data([,c("AA.cont1_La", "AA.cont1_Lb", "AA.cont1_Lc"),drop=FALSE])

The computation of the marginal totals is done, the second step is the computation of the marginal ratios.

It is done by looping on the rows of the patterns matrix

Line j of pattern is a length 3 character vector. let call patterns[j,2] x and patterns[j,3] y The programs replaces the variable names paste0("AA.cont1",x) by the ration of the variable paste0("AA.cont1",x) by the variable named paste0("AA.cont1",y).

For example for the line "3" "La_Ld_Lrn3" "La_Ld", the following replacement will be made: AA.cont1_La_Ld_Lrn3=AA.cont1_La_Ld_Lrn3/AA.cont1_La_Ld

The same is applied to all the elements of the input parameter variables.

Value

a dataframe.

Examples

```
.data=BigSyn::STtableA1
variable="AA.cont1"
variables=variable
STAtableA1<-augmentpctT_f(.data,variables)
STAtableA1$AA.cont1_[6]
STtableA1[6,names(STtableA1)[get_var(names(STtableA1))=="AA.cont1"]]
sum(STtableA1[6,names(STtableA1)[get_var(names(STtableA1))=="AA.cont1"]],na.rm=TRUE)
STtableA1[6,"AA.cont1_Lc_La_Lrn1"]
STAtableA1[6,"AA.cont1_Lc_La_Lrn1"]
```

---

augmentT_f   *Creates cell marginal max and percentages.*

Description

Creates cell marginal max and percentages.

Usage

```
augmentT_f(.data, variablesmax, variablespct, verbose = getOption("verbose"))
```

Arguments

`.data`	a dataframe
`variablesmax`	a vector of character strings
`variablespct`	a vector of character strings

Details

applys the functions augmentmaxT_f and augmentpctT_f to .data

Value

a dataframe.

Examples

```
.data=BigSyn::STtableA1
variablesmax="AA.present";variablespct="AA.cont1"
ASTtableA1<-augmentT_f(.data,variablesmax,variablespct,verbose=F)
ASTtableA1[c("AA.present_","AA.cont1_","AA.present_La","AA.cont1_La","AA.present_La_Lb","AA.cont1_La_Lb")]
```

BigSyn	*BigSyn: Some non confidential R functions developped for the MLDSC synthetic data project*

Description

The BigSyn package allows to synthesize big hierarchical databases by opposition to just a single small "rectangular" table. The general idea is to - provide tools to transpose the data and back transpose the synthetic version of the transposed data. - provide a synthetisation procedure that runs the modeling and the sampling separately - provide tools to operate a reasonable pre-selection of predictors. - provide tools to visualize the synthetisation.

BigSyn functions

The main BigSyn functions are SDPSYN2 Generaltransposefunction GeneralReversetransposefunction

General approach

NA

compilefits	*Save a pdf image of each regression tree grown in the modeling phase and discard useless information*

Description

For each element of save parameters, look at the tree and produces the corresponding pdf. It also removes all the information that is stored in the ouptut of parykit::Ctree, e.g. the data. It only keeps the tree and the rules to get it.

Usage

```
compilefits(
  Sparameters,
  fitmodelsavepath,
  pdfpath = fitmodelsavepath,
  .progress = "text"
)
```

Arguments

pdfpath	where to save the pdfs
Sparameters:	a list, that has the same structure than the outputs of
fitmodelsavepath:	
	a file path where to store the pdf of the plot
.progress:	a string, name of the progress bar to use, see plyr::create_progress_bar

Details

Depends on plyr. Partykit output contain all the data that was used to grow the tree. this function removes the unwanted information.

Examples

```
data(TtableA,package="BigSyn")
Sparameters<-Sparameters.default.f(ref.table=TtableA,asis=c("id1a","id1b"))
STtableA1<-BigSyn::SDPSYN2(TtableA,asis=c("id1a","id1b"),Sparameters=Sparameters,fitmodelsavepath = tempdir())[
pdfpath=file.path(tempdir(),"pdf")
fitmodelsavepath=tempdir()
dir.create(file.path)
 Compilefits<-compilefits(Sparameters,
 fitmodelsavepath=tempdir(),
 pdfpath=file.path(tempdir(),"pdf"))
```

---

compilesamplereports   *compilesamplereports*

Description

Sample reports are the output of the function ReportonSample

Usage

```
compilesamplereports(Sparameters, samplereportssavepath)
```

Arguments

Sparameters: a list, that has the same structure than the outputs of

samplereportssavepath:
                  a file path where to store the sample reports

Details

depends on plyr

See Also

ReportonSample

Examples

```
y=iris$Species;x=iris[,-5]
partykitctree <- partykit::ctree(y ~ ., data=cbind(y=y,x))
```

---

daniRules   *daniRules*

Description

the rules for the tree. Add an extra rule. For example, if the left branch rule is X in (a,b,c) and the right branch rule is X in (e,f), the right branch rule is modified by X not in (a,b,c). The extreme right branch rule is replaced by the negation of any of the other branches, for each node.

Usage

```
daniRules(x, i = NULL, ...)
```

Arguments

x	same format than output of partykit::ctree
i	(default: NULL)

Details

Adapted from partykit:::.list.rules.party

Value

for each terminal node of the tree, give the definition. For example: node 11 corresponds to 'x>1 and y in ("a","b")'

Examples

```
x=partykit::ctree(Petal.Width~.,iris)
daniRules(x)
```

---

donors.receptors.check
For each leave of each tree, counts the number of donors in the gold set vs the number of receptors in the synthetic dataset.

Description

For each leave of each tree, counts the number of donors in the gold set vs the number of receptors in the synthetic dataset.

Usage

```
donors.receptors.check(Rules, gold.data, syn.data)
```

Arguments

Rules	a named list (names correspond to the gold.data and syn.data variable names) of lists of logical expressions (for each variable, the logical expression corresponds to a terminal leaf in a tree).
gold.data	a data frame
syn.data	a data frame containing the same variables than gold.data

Examples

```
data(TtableA,package="BigSyn")
STtableA1=BigSyn::SDPSyn2(TtableA,asis=c("id1","id2"))
data(STtableA1,package="BigSyn")
data(package="BigSyn")
donors.receptors.check()
```

---

drop_last	*Drop last margin position (Trims all strings of a vector of strings after the last "_")*

Description

Drop last margin position (Trims all strings of a vector of strings after the last "_")

Usage

```
drop_last(x)
```

Arguments

x	a vector of character strings

Details

if x is "AA.char1_La_Ld_Lrn1" returns "AA.char1_La_Ld", if x contains no "_", returns empty string

Value

a vector of character strings

Examples

```
drop_last("AA.char1_La_Ld_Lrn1")
drop_last("iojoij")
drop_last("aa.iojoij")
```

---

fitmodel.ctree   *Function to fit a ctree model.*

Description

Function to fit a ctree model.

Usage

```
fitmodel.ctree(x, y, treeplotsavepath = NULL, ...)
```

Arguments

x   a dataframe of predictors

y   a vector :dependent variable

treeplotsavepath:
                        a path to save the graph as a pdf. if NULL, no pdf is saved

Value

a named list of 4 elements: "Rules" a data.frame with two variables: terminalnode (a integer vector)
and condition a string that gives the rule for each terminal node. "y" the values of the predictor
"terminalnodes" a vector: the terminal nodes for each element of y. "shortlist" a character string
giving the names of the variables in x that were used for the classification

Examples

```
fitmodel.ctree(x=iris[,-5],y=iris$Species)
```

---

fitmodel.fn   *Fit a model with a specific function*

Description

Fit a model with a specific function

Usage

```
fitmodel.fn(method, x, y, treeplotsavepath = NULL, ...)
```

Arguments

`method`	a string. currently only method="ctree" or "rf" (random forest).
`x`	a predictors, a dataframe.
`y`	variable to predict, a vector
`treeplotsavepath`	
	a
`...`	synthetic parameters to pass to the right fit model function. the fit model function name is the concatenation of "fit.model" and method

Value

a sublist of synparameters, which names are possible arguments of synthpop::syn.ctree if method="ctree".

Examples

```
fitmodel.fn(method="ctree",x=iris[,-5],y=iris$Species,nbuckets=30,tutu="not a good argument")
```

`fitmodel.rf`	*Function to fit a ctree model.*

Description

Function to fit a ctree model.

Usage

```
fitmodel.rf(x, y, treeplotsavepath = NULL, ...)
```

Arguments

`x`	a dataframe of predictors
`y`	a vector :dependent variable
`treeplotsavepath:`	
	a path to save the graph as a pdf. if NULL, no pdf is saved

Value

a named list of 4 elements: "Rules" a data.frame with two variables: terminalnode (a integer vector) and condition a string that gives the rule for each terminal node. "y" the values of the predictor "terminalnodes" a vector: the terminal nodes for each element of y. "shortlist" a character string giving the names of the variables in x that were used for the classification

Examples

```
fitmodel.ctree(x=iris[,-5],y=iris$Species)
```

---

fitthemodel   *Function to fit the model.*

Description

Function to fit the model.

Usage

```
fitthemodel(
  Sparameters_i,
  fitmodelsavepath,
  TtableANAto0,
  redocomputationsevenifexists = FALSE,
  treeplotsavefolder = NULL
)
```

Arguments

Sparameters_i   an element of the list output from Sparameters.default.f

fitmodelsavepath

                 a folder path. Results will either be read from or stored in this folder. If the file
                 exists, by default it is not replaced.

TtableANAto0   a table containing the predictors without NAs as well as the outcome

Value

a list.

Examples

```
#Load the data
data(TtableA,package="BigSyn")
#define parameters
Sparameters<-Sparameters.default.f(ref.table=TtableA)
Sparameters_i<-Sparameters[[53]];
fitmodelsavepath<-NULL;
TtableANAto0<-NAto0(TtableA);
redocomputationsevenifexists<-FALSE
treeplotsavefolder=tempdir()
#fit the model:
fitthemodel(Sparameters_i,fitmodelsavepath = NULL,TtableANAto0 = TtableANAto0,
            treeplotsavefolder=tempdir())
Sparameters_i<-Sparameters[["AA.present_La_La_Lrn1"]];
treeplotsavefolder=tempdir()
fitthemodel(Sparameters_i,NULL,TtableANAto0,treeplotsavefolder=tempdir())
```

GeneralReversetransposefunction

General Reverse Transpose function

Description

General Reverse Transpose function

Usage

```
GeneralReversetransposefunction(TtableA, key)
```

Arguments

key	A list of variables (columns of the transposed table)
table	A dataframe

Value

A list: first element of the list is a dataframe, the transposed version of the orioginal table. Second element is a key to allow back transposition

Examples

```
data(tableA);data(TtableA);data(XKA);key<-XKA$key
RtableA=GeneralReversetransposefunction(TtableA,key)
ordertableA <-do.call(order,tableA[c(id1,id2)])
orderRtableA<-do.call(order,RtableA[c(id1,id2)])
identical(nrow(tableA),nrow(RtableA))
identical(lapply(tableA,class),lapply(RtableA,class))
identical(tableA[ordertableA,],RtableA[orderRtableA,])
identical(names(tableA),names(RtableA))
all (lapply(names(tableA),function(x){identical(tableA[ordertableA,x],RtableA[orderRtableA,x])}))
```

GeneralReversetransposefunctiondecoupe

General Reverse Transpose function with split

Description

General Reverse Transpose function with split

Usage

```
GeneralReversetransposefunctiondecoupe(.data, key, nrowmax = 10000)
```

Arguments

`key`	A list of variables (columns of the transposed table)
`table`	A dataframe

Value

A list: first element of the list is a dataframe, the transposed version of the orioginal table. Second element is a key to allow back transposition

Examples

```
data(tableA);data(TtableA);data(XKA);key<-XKA$key
RtableA=GeneralReversetransposefunctiondecoupe(TtableA,key,10)
```

Generaltransposefunction

General Transpose function

Description

General Transpose function

Usage

```
Generaltransposefunction(
  tableA,
  id1,
  id2,
  origin = deparse(substitute(tableA))
)
```

Arguments

`id1`	A list of variables (rows)
`id2`	A list of variables (columns of the transposed table), id2 can contain as a last element the strint "rn", if the variable rn is an index for the cells formed by the variables listed first in id2
`table`	A dataframe

Value

A list: first element of the list is a dataframe, the transposed version of the orioginal table. Second element is a key to allow back transposition

Examples

```
tableA<-sampledata(TRUE)
id1=c("id1a","id1b")
id2=c("id2a","id2b")
TtableA<-Generaltransposefunction(tableA,id1,id2)
```

Generaltransposefunctionsimple

Simple General Transpose function

Description

Simple General Transpose function

Usage

```
Generaltransposefunctionsimple(tableA, id1, id2)
```

Arguments

tableA	A dataframe
id1	A list of variables (rows)
id2	A list of variables (columns of the transposed table)

Value

A data frame

Examples

```
tableA<-sampledata(TRUE)
id1=c("id1a","id1b")
id2=c("id2a","id2b")
TtableA<-Generaltransposefunctionsimple(tableA,id1,id2)
```

getnodesfromrules	*Function to get terminal node from a set of partitioning rules and new predictors*

Description

Function to get terminal node from a set of partitioning rules and new predictors

Usage

```
getnodesfromrules(x, Rules)
```

Arguments

x	a dataframe of predictors
Rules	a data frame containing 2 character variables: "terminalnode" and "condition"

Value

a vector of lenth the number of rows of x indicating the terminal nodes.

getpredictorsfromcaptureoutput	
	getpredictorsfromcaptureoutput

Description

for each terminal node of the tree, give the elements of "predictors" who appear in the node definition

Usage

```
getpredictorsfromcaptureoutput(tree, predictors)
```

Arguments

tree	same format than output of partykit::ctree
predictors	list of variables

See Also

daniRules

getpredictorsfromtree *getpredictorsfromtree*

Description

for each terminal node of the tree, give the elements of "predictors" who appear in the node definition

Usage

```
getpredictorsfromtree(tree, predictors)
```

Arguments

tree   same format than output of partykit::ctree

predictors   list of variables

See Also

daniRules

Examples

```
tree<-partykit::ctree(Petal.Width~.,iris)
getpredictorsfromtree(tree,names(iris))
```

get_cell   *get cell without the row number*

Description

get cell without the row number

Usage

```
get_cell(x, iscellrn = FALSE, iscell = FALSE)
```

Arguments

x   a vector of character strings

Details

if x is "aa.xoijj_a_1_f_1_" returns "a_1_f"

Value

a vector of character strings

Examples

```
get_cell("aa.x_1_2_3_4")#default
get_cell("1_2_3",TRUE)
get_cell("1_2_3",FALSE,TRUE)
unique(Tsampledata(TRUE)$variables))
unique(get_cell(Tsampledata(FALSE)$variables))
```

---

get_cellrn   *Get cell and row number*

Description

Get cell and row number

Usage

```
get_cellrn(x)
```

Arguments

x   a vector of character strings

Details

if x is "aa.x_a_1_f_1" returns "a_1_f_1"

Value

a vector of character strings

Examples

```
get_cellrn("AA.char1_La_Ld_Lrn1")
data(TtableA);
unique(get_cellrn(names(TtableA)))
#Second example: no transposing variables
data(TtableB);data(XKB)
unique(get_cellrn(names(XKB)))
```

get_cellXXgroup *Get cell group*

Description

Get cell group

Usage

```
get_cellXXgroup(x, marginpos, iscellXX = TRUE)
```

Arguments

x   a vector of character strings

marginpos   a vector of integer

Details

#' if x is "a_1_f_2_aa.xoijj",marginpos=2 returns "1"; if x is "a_1_f_2_aa.xoijj",marginpos=-2 returns "a_f_2"; if x is "a_1_f_2_aa.xoijj",marginpos=c(1:2) returns "a_1"

Value

a vector of character strings

Examples

```
get_cellXXgroup(c("aa.x_1_2_3_4","bb.x_1_2_3_4"),2,iscellXX=FALSE)
get_cellXXgroup(c("1_2_3_4","1_2_3_4"),2:3,iscellXX=TRUE)
variables<-Tsampledata(TRUE)$variables
unique(get_cellXXgroup(variables,2,iscellXX=FALSE))
unique(get_cellXXgroup(variables,-2,iscellXX=FALSE))
get_cellXXgroup(variables[50],2,iscellXX=FALSE)
get_cellXXgroup(variables[50],-2,iscellXX=FALSE)

#Second example: no transposing variables
TK<-Tsampledata(FALSE)
unique(get_cellXXgroup(TK$variable,1,iscell=FALSE))
```

---

get_cellXXmarginscount

Get the number of margins for a cell

Description

Get the number of margins for a cell

Usage

```
get_cellXXmarginscount(x, iscellXX = FALSE)
```

Arguments

x	a vector of character strings
iscell	a boolean indicating if x is a variable name or a cell name.

Details

if x is "aa.xoijj_a_1_f_1", cell=FALSE returns 4"; if x is "a_1_f_1", cell=TRUE returns 4"

Value

a vector of integers.

Examples

```
get_cellXXmarginscount("1_2_3_4",iscellXX=TRUE)
get_cellXXmarginscount("aa.x_1_2_3_4",iscellXX=FALSE)
data(TtableA)
unique(get_cellXXmarginscount(names(TtableA),iscellXX=FALSE))
#Second example: no transposing variables
TK<-Tsampledata(FALSE)
unique(get_cellXXmarginscount(TK$variables))
```

---

get_cellXXsplit   *split a cell*

Description

split a cell

Usage

```
get_cellXXsplit(x, marginpos = NULL, iscellXX = FALSE)
```

Arguments

x   a vector of character strings

`iscell`   x a boolean indicating if x is a cell

Details

if x is "aa.xoijj_a_1_f_1" returns c("a","1","f","1")

Value

a vector of character strings

Examples

```
get_cellXXsplit("aa.x_1_2_3_4",iscellXX=FALSE)
get_cellXXsplit("1_2_3_4",iscellXX=TRUE)
get_cellXXsplit("1_2_3_4",2:3,iscellXX=TRUE)
get_cellXXsplit("1_2_3_4",-(2:3),iscellXX=TRUE)
variables<-Tsampledata(TRUE)$variables
unique(get_cellXXsplit(variables,iscell=FALSE))
get_cellXXsplit(variables[50],iscell=FALSE)
get_cellXXsplit(variables[50],-(2:3),iscell=FALSE)
unique(get_cellXXsplit(variables,2,iscell=FALSE))
#Second example: no transposing variables
TK<-Tsampledata(FALSE)
unique(get_cellXXsplit(TK$variables,iscell=FALSE))
```

---

get_missingind   *Get missing indicator for a cell or variable*

Description

Get missing indicator for a cell or variable

Usage

```
get_missingind(x, variables)
```

Arguments

x   a vector of character strings

Details

if x is "a_1_f_1_aa.xoijj" returns c("a","1","f","1")

Value

a vector of character strings

Examples

```
variables<-Tsampledata(TRUE)$variables
unlist(unique(get_missingind(variables,variables)))
variables<-Tsampledata(FALSE)$variables
unlist(unique(get_missingind(variables,variables)))
```

---

get_natural.predictors

get variable predecessors at margin

Description

get variable predecessors at margin

Usage

```
get_natural.predictors(x, variables = x, predictors = NULL)
```

Arguments

x	a vector of character strings
variables	a vector of character strings
cells	a vector of character strings containing the potential predecessors
marginpos	a vector of integers

Details

if x is "a_1_f_1_aa.xoijj" returns c("a","1","f","1")

if x is "a_1_f_1_aa.xoijj" returns c("a","1","f","1")

Value

a vector of character strings

a vector of character strings

Examples

```
get_XXpredecessoratmargin(cellXXs="aa.x_1_2_3_4", refcellXXs=c("bb.x_1_2_2_4","aa.x_1_2_2_4","aa.x_1_1_3_4"),2,
get_XXpredecessoratmargin(cellXXs=c("1_2_2_4","1_2_2_4","1_1_3_4","1_1_3_3"),iscellXX=FALSE)
data(XKA)
cells<-unique(get_cellrn(XKA$variables))
get_XXpredecessoratmargin(cells,marginpos=1,iscellXX=TRUE)
get_XXpredecessoratmargin(cells[10],cells,1,iscellXX=TRUE)
Get natural predictors

TK<-TtableA
get_natural.predictors(x=sample(names(TtableA),5),variables=names(TtableA))
```

---

get_origin   *Get origin table*

Description

Get origin table

Usage

```
get_origin(x)
```

Arguments

x   a vector of character strings

Details

if x is "aa.xoijj_a_1_f_1_" returns c("aa")

Value

a vector of character strings

Examples

```
get_origin("tableA.cont1_1_Lrn1")
variables<-Tsampledata(TRUE)$variables
unlist(unique(get_origin(variables,variables)))
variables<-Tsampledata(FALSE)$variables
unlist(unique(get_origin(variables,variables)))
```

---

get_parent   *get parent if any*

Description

get parent if any

Usage

```
get_parent(variables, variable_ref)
```

Arguments

variables   a character strings

variable_ref   a vector of character strings

Details

if x is "aa.xoijj_a_1_f_1_" returns "a_1_f"

Value

a vector of character strings

Examples

```
get_parent("aa.x_1_2_3_4","aa.x_1_2_3")#default
```

---

get_presentind   *get the present indicator for a cell*

Description

get the present indicator for a cell

Usage

```
get_presentind(
  variables,
  refvariables = variables,
  rns = unlist(unique(get_cellrn(refvariables)))
)
```

Arguments

x   a vector of character strings

Details

if x is "a_1_f_1_aa.xoijj" returns c("a","1","f","1")

Value

a vector of character strings

Examples

```
get_presentind("AA.x_1_2_3_4","AA.present_1_2_3_4")
get_presentind("AA.present_1_2_3_4",c("AA.present_1_2_3_3","AA.present_1_2_3"))
get_presentind("AA.present_1_2_3_4",c("AA.present_1_2_3_3","AA.present_1_2_3_4"))
variables<-Tsampledata(TRUE)$variables
variable<-"AA.present_La_La_Lrn1"
get_presentind(variable,variables)
unlist(unique(get_presentind(variables)))
variables<-Tsampledata(FALSE)$variables
unlist(unique(get_presentind(variables,variables)))
```

---

get_var   *Get variable name*

Description

Get variable name

Usage

```
get_var(x)
```

Arguments

x   a vector of character strings

Details

if x is "aa.xoijj_a_1_f_1" returns "aa.xoijj"

Value

a vector of character strings

Examples

```
get_var("aa.x_1_2_3_4")
data(TtableA)
unique(get_var(names(TtableA)))
#Second example: no transposing variables
TK<-Tsampledata(FALSE)
unique(get_var(TK$variables))
```

---

get_XXpredecessoratmargin

                          get cell predecessors at margin

Description

get cell predecessors at margin

Usage

```
get_XXpredecessoratmargin(
  XXs,
  refXXs = XXs,
  marginpos = NULL,
  iscellXX = FALSE
)
```

Arguments

`XXs`	a vector of character strings
`refXXs`	a vector of character strings containing the potential predecessors
`marginpos`	a vector of integers

Details

if XXs is "aa.xoijj_a_1_f_1" and refXXs contains "aa.xoijj_a_1_e_1" and marginpos=3 returns "aa.xoijj_a_1_e_1" if XXs is "aa.xoijj_a_1_f_2" and refXXs contains "aa.xoijj_a_1_f_1" and marginpos=NULL returns "aa.xoijj_a_1_f_1" if XXs is "id1" and iscellXX=FALSE whatever refXXs returns character(0) if XXs is "" and iscellXX=FALSE whatever refXXs returns character(0) if XXs is "b_1_f_1" and iscellXX=TRUE and refXXs contains "a_1_f_1" returns "a_1_f_1"

Value

a vector of character strings

Examples

```
get_XXpredecessoratmargin(XXs="aa.x_1_2_3_4", refXXs=c("bb.x_1_2_2_4","aa.x_1_2_2_4","aa.x_1_1_3_4"),2,iscellX)
get_XXpredecessoratmargin(XXs=c("1_2_2_4","1_2_2_4","1_1_3_4","1_1_3_3"),iscellXX=TRUE)
get_XXpredecessoratmargin(XXs="1_1_3_4",refXXs=c("1_2_2_4","1_2_2_4","1_1_3_4","1_1_3_3"),iscellXX=TRUE)
data(XKA)
cells<-unique(get_cellrn(XKA$variables))
get_XXpredecessoratmargin(cells,marginpos=1,iscellXX=TRUE)
get_XXpredecessoratmargin(cells[10],cells,1,iscellXX=TRUE)
```

---

get_XXpredecessorsatmargin

get cell predecessors at margin

Description

get cell predecessors at margin

Usage

```
get_XXpredecessorsatmargin(
  XXs,
  marginpos,
  refXXs = XXs[order(get_cellXXgroup(XXs, marginpos))],
  iscellXX = FALSE,
  cellXXgroup = get_cellXXgroup(refXXs, marginpos2, iscellXX),
  CompcellXXgroup = get_cellXXgroup(refXXs, -marginpos2, iscellXX)
)
```

Arguments

XXs	a vector of character strings
marginpos	a vector of integers
refXXs	a vector of character strings containing the potential predecessors

Details

if XXs is "aa.xoijj_a_1_f_1" and refXXs contains "aa.xoijj_a_1_e_1" and marginpos=3 returns "aa.xoijj_a_1_e_1" if XXs is "aa.xoijj_a_1_f_2" and refXXs contains "aa.xoijj_a_1_f_1" and marginpos=NULL returns "aa.xoijj_a_1_f_1" if XXs is "id1" and iscellXX=FALSE whatever refXXs returns character(0) if XXs is "" and iscellXX=FALSE whatever refXXs returns character(0) if XXs is "b_1_f_1" and iscellXX=TRUE and refXXs contains "a_1_f_1" returns "a_1_f_1"

Value

a vector of character strings

Examples

```
get_XXpredecessoratmargin(XXs="aa.x_1_2_3_4", refXXs=c("bb.x_1_2_2_4","aa.x_1_2_2_4","aa.x_1_1_3_4"),2,iscellX
get_XXpredecessoratmargin(XXs=c("1_2_2_4","1_2_2_4","1_1_3_4","1_1_3_3"),iscellXX=TRUE)
get_XXpredecessoratmargin(XXs="1_1_3_4",refXXs=c("1_2_2_4","1_2_2_4","1_1_3_4","1_1_3_3"),iscellXX=TRUE)
data(XKA)
cells<-unique(get_cellrn(XKA$variables))
get_XXpredecessoratmargin(cells,marginpos=1,iscellXX=TRUE)
get_XXpredecessoratmargin(cells[10],cells,1,iscellXX=TRUE)
```

---

good.fit.parameters	*Select only arguments for a specific fitting function*

Description

Select only arguments for a specific fitting function

Usage

```
good.fit.parameters(method, synparameters)
```

Arguments

method	a string. currently only method="ctree".
synparameters	a named list.

Details

Currently only works with method="ctree" Only selects the arguments that match the function partykit::ctree_control

Value

a sublist of synparameters, which names are possible arguments of partykit::ctree_control if method="ctree".

Examples

```
good.fit.parameters(method="ctree",list(teststat=30,tutu=3))
```

```
good.sample.parameters
```
                                    Select only arguments for sampling.

Description

Select only arguments for sampling.

Usage

```
good.sample.parameters(method, synparameters)
```

Arguments

method	a string. currently only method="ctree".
synparameters	a named list.

Details

In prevision of future developments. returns NULL for the moment

Examples

```
good.sample.parameters(method="ctree",list(teststat=30,tutu=3))
```

```
good.syn.parameters
```
                    Select only arguments for a specific fitting function This function is not used anymore

---

Description

Select only arguments for a specific fitting function This function is not used anymore

Usage

```
good.syn.parameters(method, synparameters)
```

Arguments

`method`	a string. currently only method="ctree".
`synparameters`	a named list.

Details

Currently only works with method="ctree" Only selects the arguments that match the function synthpop::syn.ctree

Value

a sublist of synparameters, which names are possible arguments of synthpop::syn.ctree if method="ctree".

Examples

```
good.syn.parameters(method="ctree",list(y=c(1:2),smoothing=TRUE,tutu="not in synthpop::syn.ctree arguments"))
```

NAto0	*Recoding of NAs to 0 or "NA"*

Description

Recoding of NAs to 0 or "NA"

Usage

```
NAto0(tableA)
```

Arguments

`tableA`	a dataframe

Details

for synthetisation to run, missing values are treated as a special factor level for factor variables, or as 0 for continuous variables. To avoid issues, for continuous variables, a missing indicator is also created.

Value

a dataframe

Examples

```
toto<-cars
toto$speed[sample(nrow(cars),3)]<-NA
NAto0(toto)
```

| onlygoodargs | *Generic function: remove all the elements of a named list which names are not arguments of a specific function.* |

Description

Generic function: remove all the elements of a named list which names are not arguments of a specific function.

Usage

```
onlygoodargs(fun, L)
```

Arguments

fun	a function
L	a list

Details

remove all the elements of a list which names are not arguments of a specific function.

Value

a list.

Examples

```
onlygoodargs(lm,list(data=cars,formula=speed~dist,tutu="not an arg from lm"))
```

| posixcttonumeric | *Converts all posixct variables of a dataframe into a numeric variable* |

Description

Converts all posixct variables of a dataframe into a numeric variable

Usage

```
posixcttonumeric(tableA)
```

Arguments

tableA	a dataframe

Value

a list.

Examples

```
toto<-cars
toto$now<-Sys.time()
posixcttonumeric(toto)
```

```
predictor.matrix.default
```
Define a default predictor matrix

Description

Define a default predictor matrix

Usage

```
predictor.matrix.default(variables)
```

Arguments

variables   a vector of character strings

Details

Returns the lower diagonal matrix with ones.

Value

a matrix

Examples

```
variables<-Tsampledata(TRUE)$variables
predictor.matrix.default(TK$variables)
```

---

predictor.matrix.rate   *predictor.matrix.rate*

Description

predictor.matrix.rate

Usage

```
predictor.matrix.rate(
  variables,
  nopredictor = character(0),
  allpredictor = character(0),
  marginposs = integer(0)
)
```

Arguments

x   a vector of character strings

Details

if x is "aa.xoijj_a_1_f_1_" returns c("a","1","f","1")

Value

a vector of character strings

---

preparepredictorsforctreefit
                                  Prepare predictors for ctree fit

---

Description

Prepare predictors for ctree fit

Usage

```
preparepredictorsforctreefit(x, keep = NULL)
```

Arguments

x   a predictors, a dataframe.

method   a string. currently only method="ctree".

y   variable to predict, a vector

...   synthetic parameters to pass to the right fit model function. the fit model function
                     name is the concatenation of "fit.model" and method

Value

a sublist of synparameters, which names are possible arguments of synthpop::syn.ctree if method="ctree".

Examples

```
preparepredictorsforctreefit(x,
                             keep=NULL)
```

| reduceT_f | *Reverse augmentT_f: Function that will convert cell and marginal ra-tios and overall total to cell values* |

Description

Reverse augmentT_f: Function that will convert cell and marginal ratios and overall total to cell values

Usage

```
reduceT_f(.data, variables, verbose = FALSE, hack = TRUE)
```

Arguments

.data	data frame to "reduce"
variables	list of variable names roots
verbose	(default FALSE) if verbose, the formulae to compute the new variables is printed.
hack	(default TRUE)

Details

this functions looks for the Augmentation parameters in the package object Augmentparameters[[tablename]]$percent For each variable listed in Augmentparameters[[tablename]]$percent, it looks for the corresponding variable in .data and computes cell values from cell and marginal ratios and overall total

---

resampleT_f	*resample when synthetisation created incoherent high and low level aggregates*

Description

resample when synthetisation created incoherent high and low level aggregates

Usage

```
resampleT_f(.data, variables, verbose = FALSE)
```

Arguments

.data	data frame to "reduce"
variables	list of variable names roots
verbose	(default FALSE) if verbose, the formulae to compute the new variables is printed.

---

runCompare	*runCompare*

Description

Shiny App to visualize regression trees and compare synthetic vs non synthetic data

Usage

```
runCompare(
  data1 = NULL,
  data2 = NULL,
  listofpackage1 = installed.packages()[, "Package"],
  listofpackage2 = installed.packages()[, "Package"],
  package1 = if (is.element("BigSyn", listofpackage1)) {     "BigSyn" } else {
    listofpackage1[1] },
  package2 = if (is.element("BigSyn", listofpackage2)) {     "BigSyn" } else {
    listofpackage2[1] }
)
```

Arguments

data1	a dataframe
data2	a dataframe
listofpackage1	a vector of character strings
listofpackage2	a vector of character strings

```
package1        a character string

package2        a character string

Sparameters
```

Examples

```
package1<-NULL
package2<-NULL
runCompare()
```

---

sample.ctree	*Function to sample from a ctree fitted model*

Description

Function to sample from a ctree fitted model

Usage

```
sample.ctree(xp, fit.model, smoothing = "none", ...)
```

Arguments

```
y                 a vector of values to pull from

terminalnodes     a vector of terminal nodes
newterminalnodes:
                  a path to save the graph
```

Value

a vector of the same size than terminalnodes, obtained by sampling betweenn the values of y such for the same terminal node.

---

sample.fn	*Sample a model with a specific function*

Description

Sample a model with a specific function

Usage

```
sample.fn(method, xp, fit.model, smoothing, ...)
```

Arguments

method	a string. currently only method="ctree".
...	synthetic parameters to pass to the right fit model function. the fit model function name is the concatenation of "fit.model" and method
x	a predictors, a dataframe.
y	variable to predict, a vector

Value

a sublist of synparameters, which names are possible arguments of synthpop::syn.ctree if method="ctree".

Examples

```
sample.fn(method="ctree",
          xp=iris[,-5],
          fit.model=fitmodel.fn(method="ctree",x=iris[,-5],y=iris$Species,nbuckets=30),
          smoothing=FALSE)
```

---

sampledata	*Sample data for transposition*

Description

Sample data for transposition

Usage

```
sampledata(transposingvariables = TRUE)
```

Arguments

transposingvariables
> a boolean. If TRUE, transposing id variables are created.

Value

a data frame with id variables, numeric, factor and character variables.

samplefrompool

Function to sample from a set of partitioning rules

Description

Function to sample from a set of partitioning rules

Usage

```
samplefrompool(y, terminalnodes, newterminalnodes)
```

Arguments

y   a vector of values to pull from

terminalnodes   a vector of terminal nodes

newterminalnodes:
                 a path to save the graph

Value

a vector of the same size than terminalnodes, obtained by sampling betweenn the values of y such for the same terminal node.

Examples

```
y=iris$Species;x=iris[,-5];fit.mod<-fitmodel.ctree(x,y);terminalnodes<-getnodesfromrules(x,fit.mod$Rules);
newterminalnodes<-sample(unique(terminalnodes),10,replace=TRUE);
samplefrompool(y,terminalnodes,newterminalnodes)
y<-y[terminalnodes!=7]
terminalnodes<-terminalnodes[terminalnodes!=7]
samplefrompool(y,terminalnodes,newterminalnodes)
```

SDPSYN2

General SDP function.

Description

General SDP function.

Usage

```
SDPSYN2(
  TtableA,
  asis = NULL,
  notpredictor = asis,
  nrep = 1,
  synparameters = NULL,
  Sparameters = Sparameters.default.f(ref.table = TtableA, asis = asis, notpredictor =
    notpredictor, preferredmethod = "ctree", defaultsynparameters =
    c(as.list(synparameters),
   eval(formals(Sparameters.default.f)$defaultsynparameters)[setdiff(names(formals(Sparameters.defau
    c("", names(synparameters)))])),
  STtableA = if (is.null(asis)) {    data.frame(.n = rep(nrep, each = nrow(TtableA))) }
   else {    plyr::ddply(data.frame(.n = nrep), ~.n, function(d) {       TtableA[asis]
        }) },
  fitmodelsavepath = NULL,
  treeplotsavefolder = NULL,
  samplereportsavepath = NULL,
  stepbystepsavepath = NULL,
  doparallel = TRUE,
  recode = NULL,
  saveeach = 200,
  randomfitorder = TRUE,
  fitonly = FALSE
)
```

Arguments

TtableA	a dataframe to synthesize
asis	list of variable names from TtableA to keep as is (e.g. not to synthesize)
notpredictor	list of variable names which should not be used as predictors
nrep	number of synthetic replicates wanted
synparameters	general synthetisation paramters
Sparameters	a list, Specific (variable by variable) synthetisation parameters, splits ...
STtableA	a dataframe
fitmodelsavepath	
	a path where to save the fitted models
treeplotsavefolder	
	a path where to save the tree plots
samplereportsavepath	
	a path where to save the sampling report
stepbystepsavepath	
	a path where to backup the synthetised in case of a crash
doparallel	a boolean indicating whether sampling should be done in parallel for each replicacte

recode	: a vector of character strings or NULL, list of variables to be recoded
saveeach	an integer, indicating every how many variables a backup is done
randomfitorder	a boolean : fitting for each variable can be done in the order of appearance of each variables or at random
fitonly	a boolean, if TRUE, no sampling is done.

Details

This function is doing both the fitting and the sampling.

Examples

```
data(TtableA,package="BigSyn")
ATtableA=augmentT_f(TtableA,variablesmax="AA.present",variablespct="AA.cont1")
asis=NULL;notpredictor=asis;nrep=1;synparameters=NULL;
Sparameters=Sparameters.default.f(ref.table=TtableA,asis=asis,notpredictor=notpredictor,preferredmethod="ctree
defaultsynparameters=c(as.list(synparameters),
eval(formals(Sparameters.default.f)$defaultsynparameters))[setdiff(names(formals(Sparameters.default.f)$default
SATtableA=plyr::rdply(nrep,ATtableA[asis]);samplereportsavepath=NULL;stepbystepsavepath=NULL;doparallel=FALSE;
fitmodelsavepath=tempdir()
treeplotsavefolder=tempdir()
sapply(list.files(tempdir(),full.names = TRUE  ),file.remove)
SATtableA<-SDPSYN2(ATtableA,asis=NULL,fitmodelsavepath = fitmodelsavepath,treeplotsavefolder=treeplotsavefolder
todisplay<-grep("La_La_Lrn1",names(STtableA[[1]]),value=T);
STtableA[[1]][1:3,todisplay];TtableA[1:3,todisplay]
############################################################
Controling that AA.present_La=0=>AA.present_La_Lb=0 in synthetic data
library(BigSyn)
library(reshape2)
library(data.table)
data(TtableA,package="BigSyn")
variablepct="AA.cont1"
variablespct=variablepct
variablemax="AA.present"
variablesmax=variablemax
set.seed(1)
asis=c("id1a", "id1b")
                fitmodelsavepath=NULL
                treeplotsavefolder=NULL
                samplereportsavepath=NULL
                stepbystepsavepath=NULL
                doparallel=TRUE
                recode=NULL
                saveeach=200
                randomfitorder=TRUE
                fitonly=FALSE

variablemax="AA.present"
variablesmax=variablemax
variablepct="AA.cont1"
```

```
variablespct=variablepct
ATtableA<-augmentT_f(TtableA,variablesmax=variablesmax,variablespct=variablespct)
TtableA<-ATtableA
STtableA<-ATtableA[asis]
Sparameters=Sparameters.default.f(ref.table=ATtableA,asis=c("id1a", "id1b"),
    notpredictor=NULL,
    preferredmethod="ctree",
    defaultsynparameters=eval(formals(Sparameters.default.f)$defaultsynparameters))
SATtableA<-BigSyn::SDPSYN2(ATtableA,asis=c("id1a", "id1b"))[[1]]
problems<-SATtableA$AA.present_Lb_La==1&SATtableA$AA.present_Lb==0
mean(problems)
Sparameters[["AA.present_Lb_La"]]
library(dplyr)
library(ggplot2)
xx<-function(x){
xxx<-x[sort(grep("present",names(x),value=TRUE))]
xxx[xxx==0]<-NA
StudyDataTools::ggplot_missing(xxx)}
xx(ATtableA)
xx(SATtableA)
```

---

sorttablewithingroup   *sort table within group without changing the group position*

Description

sort table within group without changing the group position

Usage

```
sorttablewithingroup(.data, groupvar, sortvar, decreasing = FALSE)
```

Arguments

.data	a dataframe
groupvar	a vector of character strings that are names of variables from .data
sortvar	a vector of character strings that are names of variables from .data
decreasing	a boolean if TRUE, decreasing order is used.

Details

Groups are defined by unique values of .data[groupvar]. Within each group, data is sorted according to sortvar.

Value

a list.

Examples

```
set.seed(1)
N=10
.data=data.frame(
 .group=sample(letters[1:2],N,replace=TRUE),
 y=runif(N),
 origorder=1:N)
 groupvar=".group"
 sortvar="y"
.data2=plyr::ddply(.data,".group",function(d){d$intraorder=order(d[[sortvar]]);d$neworder=d$origorder[order(or
.data4=.data2[.data2[[groupvar]]=="a",];.data4[.data4$intraorder,]
cbind(.data,"|",.data2)
cbind(.data,"|",.data2[order(.data2$origorder),])
cbind(.data[order(order(.data2$origorder)),],"|",.data2)
.data3=cbind(.data,"|",.data2[order(.data2$neworder),]);.data3[.data[[groupvar]]=="a",];.data3
.data3=cbind(.data,"|",.data[order(order(.data2$origorder)),][order(.data2$neworder),]);.data3[.data[[groupvar
.data3=cbind(.data,"|",.data[order(order(.data2$origorder))[order(.data2$neworder)],]);.data3[.data[[groupvar]
cbind(.data,I="|",sorttablewithingroup(.data,groupvar,sortvar),I="|",sorttablewithingroup(.data,groupvar,sortv
```

Sparameters.default.f   *Default synthetisation parameters based on variable names*

Description

Default synthetisation parameters based on variable names

Usage

```
Sparameters.default.f(
  ref.table,
  asis = NULL,
  notpredictor = NULL,
  variables = Sparameters.variables.reorder.default(names(ref.table)),
 predictors.matrix = predictor.matrix.default(variables)[!is.element(variables, asis),
    !is.element(variables, notpredictor)],
  splittingvar = NULL,
  moresplits = NULL,
  preferredmethod = "ctree",
  splithreshold = 100,
 defaultsynparameters = list(smoothing = "none", importance = TRUE, keep.forest = TRUE,
    minbucket = 30)
)
```

Arguments

ref.table	a dataframe
asis	a vector of character strings, indicating which variables to keep as is.

notpredictor   a vector of character strings, indicating which variables are not supposed to be
                    used as predictors.

variables   a vector of character strings, indicating the variables to synthesize. Order is
                    important.

predictors.matrix

                    a predictor matrix. Number of rows is the number of variables to synthesize,
                    number of columns is all the variables from ref.table

moresplits   an object of class moresplist (not defined yet)

preferredmethod:

                    "rf" for random forest or "ctree" for classification tree

defaultparameters

                    a list indicating default parameters for synthpop synthesisation functions, for
                    example ntree=5, smoothing="none"

Details

creates default synthetisation parameters Some rules: parents variable are potential predictors of
their children, synthetisation is conditional to missingindicators, synthetisation is conditional to
presence in cell

Examples

```
data(TtableA)
ATtableA<-augmentT_f(TtableA,variablespct="AA.cont1",variablesmax="AA.present")
ref.table<-ATtableA
Spa<-Sparameters.default.f(ref.table=ATtableA)
names(Spa)<-lapply(Spa,function(x){x$variable})
Spa$AA.present_La_Lb
Spa$AA.cont1_La_Lb
```

Sparameters.variables.reorder.default
                    General Default ordering of variables for synthetisation based on
                    name of the variable.

Description

General Default ordering of variables for synthetisation based on name of the variable.

Usage

```
Sparameters.variables.reorder.default(
  variables,
  orderwithinorigin = NULL,
  id = NULL,
  extrasort = NULL
)
```

Arguments

variables   vector of character strings, indicating names of variables

orderwithinorigin
                 a list, see example

id   a vector of character strings

Details

After transposition, variable names follow this format: origin.variablename_margin1_margin2....lastmargin
Some rules have to be followed:

- Missing indicators have to be synthesised before the corresponding variables, for example AA.factor1missingind_L1_L2_L1
needs to be synthesised before AA.factor1missingind_L1_L2_L1

- Cell indicators must be synthesised before the corresponding variables. For example AA.present_L1_L2_L1
must be synthesised before AA.factor1_L1_L2_L1 and before AA.cont1_L1_L2_L1

- Parent variables (aggregated) must be synthesised before their children: For example AA.present_L1
must be synthesised before AA.present_L1_L2, AA.cont2_L1_L2 must be synthesised before AA.cont2_L1_L2_L3
AA.present_L1 must be synthesised before AA.present_L1_L3 AA.cont2missingind_L1 must be
synthesised before AA.cont2missingind_L1_L3

- if for examples variable AA.cont1 in each cell has to be synthesised before AA.cont2, this can be
specified with the orderwithinorigin argument

- for the use of the argument extrasort, refer to sorttablewithingroup

Value

a list.

Examples

```
TK<-Tsampledata(TRUE)$TtableA
Sparameters.variables.reorder.default(names(TK$TtableA))
#Second example: no transposing variables
TtableA<-Tsampledata(TRUE)$TtableA
orderwithinorigin=c("AA.factor1","AA.factor2")
variables<-names(TtableA)
Sparameters.variables.reorder.default(variables,orderwithinorigin)
```

---

treedepth   *Compute depth of a "party" tree*

Description

this function takes the output of the partykit::ctree function and prints the tree into a pdf file, located
in the specified folder. It computes the depth and width and tries to create a pdf with the right
dimensions.

Usage

```
treedepth(x)
```

Arguments

x   a tree

Details

recursive function

Examples

```
y=iris$Species;x=iris[,-5]
partyctree <- party::ctree(y ~ ., data=cbind(y=y,x))
treedepth(partyctree@tree)
partyctree <- party::ctree(y ~ ., data=cbind(y=y,x))
treedepth(partykit::ctree(y ~ ., data=cbind(y=y,x)))
```

---

treetopdf   *Ctree to pdf graph*

Description

this function takes the output of the partykit::ctree function and prints the tree into a pdf file, located in the specified folder. It computes the depth and width and tries to create a pdf with the right dimensions.

Usage

```
treetopdf(partykitctree, savepath)
```

Arguments

partykitctree: an output of partykit::ctree

savepath: a file path where to store the pdf of the plot

Examples

```
y=iris$Species;x=iris[,-5]
partykitctree <- partykit::ctree(y ~ ., data=cbind(y=y,x))
treetopdf(partykitctree,"./x.pdf")
```

Tsampledata   *Transposed sample data.*

Description

Transposed sample data.

Usage

```
Tsampledata(transposingvariables = TRUE)
```

Arguments

```
transposingvariables
```
      a boolean. If TRUE, stransposing id variables are created.

Details

Tsampledata(x) is Generaltransposefunction(Tsampledata(x))

Value

a data frame with id variables, numeric, factor and character variables.

TTsampledata   *Transposed sample data.*

Description

Transposed sample data.

Usage

```
TTsampledata(transposingvariables = TRUE)
```

Arguments

```
transposingvariables
```
      a boolean. If TRUE, stransposing id variables are created.

Details

Tsampledata(x) is Generaltransposefunction(Tsampledata(x))

Value

a data frame with id variables, numeric, factor and character variables.

`%notin%`   *' ... %notin% NA negation de ' 1%notin%2:3 1%notin%1:3*

Index