

Package ‘vtreat’

October 25, 2016

Type Package

Title A Statistically Sound data.frame Processor/Conditioner

Version 0.5.28

Date 2016-10-24

Author John Mount, Nina Zumel

URL <https://github.com/WinVector/vtreat>

BugReports <https://github.com/WinVector/vtreat/issues>

Maintainer John Mount <jmount@win-vector.com>

Description A data.frame processor/conditioner that prepares real-world data for predictive modeling in a statistically sound manner. Prepares variables so that data has fewer exceptional cases, making it easier to safely use models in production. Common problems 'vtreat' defends against: Inf, NA, too many categorical levels, rare categorical levels, new categorical levels (levels seen during application, but not during training). 'vtreat::prepare' should be used as you would use 'model.matrix'.

License GPL-3

Suggests testthat, knitr, parallel, rmarkdown, dplyr, ggplot2

LazyData true

VignetteBuilder knitr

RoxygenNote 5.0.1

NeedsCompilation no

Repository CRAN

Date/Publication 2016-10-25 00:18:32

R topics documented:

vtreat-package	2
buildEvalSets	4
catScore	6
designTreatmentsC	6

designTreatmentsN	8
designTreatmentsZ	10
format.vtreatment	11
getSplitPlanAppLabels	11
kWayCrossValidation	12
kWayStratifiedY	13
linScore	14
makekWayCrossValidationGroupedByColumn	14
mkCrossFrameCExperiment	15
mkCrossFrameNExperiment	17
oneWayHoldout	18
prepare	19
print.vtreatment	20
problemAppPlan	21
vnames	21
vorig	22

Index 23

vtreat-package	<i>vtreat: a package for simple variable treatment</i>
----------------	--

Description

vtreat is a data.frame processor/conditioner that prepares real-world data for predictive modeling in a statistically sound manner.

Common problems vtreat defends against: NA, too many categorical values, rare categorical values, new categorical values (levels seen during application, but not during training). `prepare` should be used as you would use `model.matrix`. `prepare` treated variables are all numeric and never take the value NA (so are very safe to use in modeling).

To use: first build a list of variable treatments from your training data using `designTreatmentsC` (for models predicting binary categorical outcomes) or `designTreatmentsN` (for models predicting numeric outcomes). If you have enough data we suggested running the design step on a subset of data disjoint from training and test (this avoids many issues including mis-counting degrees of freedom on effect or impact codes arising from categorical variables with a large number of levels). Then apply a list of treatments to a data frame to get a treated data frame using `prepare`. All the code is assuming we are working only with rows where the outcome or y-value is not NA, finite and not nan.

Details

```
Package: vtreat
Type: Package
Version: 0.5.28
Date: 2016-10-24
License: GNU General Public License version 3
```

Author(s)

John Mount and Nina Zumel

Maintainer: <contact@win-vector.com>

References

See: <http://www.win-vector.com/blog/2014/06/r-minitip-dont-use-data-matrix-when-you-mean-model-matrix/> <http://www.win-vector.com/blog/2012/07/modeling-trick-impact-coding-of-categorical-variables-with-many-levels/> <http://practicaldatascience.com/> "Effect codes" in Cohen et. al. Applied multiple regression/correlation for the behavioral sciences.

See Also

For more information:

- `vignette(package='vtreat')`
- `RShowDoc('vtreat', package='vtreat')`
- `RShowDoc('vtreatVariableTypes', package='vtreat')`
- `RShowDoc('vtreatOverfit', package='vtreat')`
- Website: <https://github.com/WinVector/vtreat>

Examples

```
# categorical example
dTrainC <- data.frame(x=c('a','a','a','b','b',NA,NA),
  z=c(1,2,3,4,NA,6,NA),y=c(FALSE,FALSE,TRUE,FALSE,TRUE,TRUE,TRUE))
dTestC <- data.frame(x=c('a','b','c',NA),z=c(10,20,30,NA))
treatmentsC <- designTreatmentsC(dTrainC,colnames(dTrainC),'y',TRUE)
dTrainCTreated <- prepare(treatmentsC,dTrainC,pruneSig=1.0,scale=TRUE)
varsC <- setdiff(colnames(dTrainCTreated),'y')
# all input variables should be mean 0 for significant variables
sapply(dTrainCTreated[,varsC,drop=FALSE],mean)
# all non NA slopes should be 1
sapply(varsC,function(c) { glm(paste('y',c,sep='~'),family=binomial,
  data=dTrainCTreated)$coefficients[[2]])})
dTestCTreated <- prepare(treatmentsC,dTestC,pruneSig=c(),scale=TRUE)

# numeric example
dTrainN <- data.frame(x=c('a','a','a','a','b','b',NA,NA),
  z=c(1,2,3,4,5,NA,7,NA),y=c(0,0,0,1,0,1,1,1))
dTestN <- data.frame(x=c('a','b','c',NA),z=c(10,20,30,NA))
treatmentsN = designTreatmentsN(dTrainN,colnames(dTrainN),'y')
dTrainNTreated <- prepare(treatmentsN,dTrainN,pruneSig=1.0,scale=TRUE)
varsN <- setdiff(colnames(dTrainNTreated),'y')
# all input variables should be mean 0
sapply(dTrainNTreated[,varsN,drop=FALSE],mean)
# all non NA slopes should be 1
```

```
sapply(varsN,function(c) { lm(paste('y',c,sep='~'),
  data=dTrainNTreated)$coefficients[[2]])
dTestNTreated <- prepare(treatmentsN,dTestN,pruneSig=c(),scale=TRUE)

# for large data sets you can consider designing the treatments on
# a subset like so: d[sample(1:dim(d)[[1]],1000),,drop=FALSE]
```

 buildEvalSets

Build set carve-up for out-of sample evaluation.

Description

Return a carve-up of `seq_len(nRows)`. Very useful for any sort of nested model situation (such as data prep, stacking, or super-learning).

Usage

```
buildEvalSets(nRows, ..., dframe = NULL, y = NULL, splitFunction = NULL,
  nSplits = 3)
```

Arguments

<code>nRows</code>	scalar, ≥ 1 number of rows to sample from.
<code>...</code>	no additional arguments, declared to forced named binding of later arguments.
<code>dframe</code>	(optional) original data.frame, passed to user <code>splitFunction</code> .
<code>y</code>	(optional) numeric vector, outcome variable (possibly to stratify on), passed to user <code>splitFunction</code> .
<code>splitFunction</code>	(optional) function taking arguments <code>nSplits</code> , <code>nRows</code> , <code>dframe</code> , and <code>y</code> ; returning a user desired split.
<code>nSplits</code>	integer, target number of splits.

Details

Also sets attribute "splitmethod" on return value that describes how the split was performed. `attr(returnValue,'splitmethod')` is one of: 'notsplit' (data was not split; corner cases like single row data sets), 'oneway' (leave one out holdout), 'kwaycross' (a simple partition), 'userfunction' (user supplied function was actually used), or a user specified attribute. Any user desired properties (such as stratification on `y`, or preservation of groups designated by original data row numbers) may not apply unless you see that 'userfunction' has been used.

The intent is the user `splitFunction` only needs to handle "easy cases" and maintain user invariants. If the user `splitFunction` returns `NULL`, throws, or returns an unacceptable carve-up then `vtreat::buildEvalSets` returns its own eval set plan. The signature of `splitFunction` should be `splitFunction(nRows,nSplits,dframe,y)` where `nSplits` is the number of pieces we want in the carve-up, `nRows` is the number of rows to split, `dframe` is the original dataframe (useful for any group control variables), and `y` is a numeric vector representing outcome (useful for outcome stratification).

Note that `buildEvalSets` may not always return a partition (such as one row dataframes), or if the user split function chooses to make rows eligible for application a different number of times.

Value

list of lists where the app portion of the sub-lists is a disjoint carve-up of seq_len(nRows) and each list as a train portion disjoint from app.

See Also

[kWayCrossValidation](#), [kWayStratifiedY](#), and [makekWayCrossValidationGroupedByColumn](#)

Examples

```
# use
buildEvalSets(200)

# longer example
# helper fns
# fit models using experiment plan to estimate out of sample behavior
fitModelAndApply <- function(trainData, applicaitonData) {
  model <- lm(y~x, data=trainData)
  predict(model, newdata=applicaitonData)
}
simulateOutOfSampleTrainEval <- function(d, fitApplyFn) {
  eSets <- buildEvalSets(nrow(d))
  evals <- lapply(eSets,
    function(ei) { fitApplyFn(d[ei$train,], d[ei$app,]) })
  pred <- numeric(nrow(d))
  for(eii in seq_len(length(eSets))) {
    pred[eSets[[eii]]$app] <- evals[[eii]]
  }
  pred
}

# run the experiment
set.seed(2352356)
# example data
d <- data.frame(x=rnorm(5), y=rnorm(5),
  outOfSampleEst=NA, inSampleEst=NA)

# fit model on all data
d$inSampleEst <- fitModelAndApply(d, d)
# compute in-sample R^2 (above zero, falsely shows a
# relation until we adjust for degrees of freedom)
1-sum((d$y-d$inSampleEst)^2)/sum((d$y-mean(d$y))^2)

d$outOfSampleEst <- simulateOutOfSampleTrainEval(d, fitModelAndApply)
# compute out-sample R^2 (not positive,
# evidence of no relation)
1-sum((d$y-d$outOfSampleEst)^2)/sum((d$y-mean(d$y))^2)
```

catScore	<i>return significnace 1 variable logistic regression</i>
----------	---

Description

return significnace 1 variable logistic regression

Usage

```
catScore(varName, x, yC, yTarget, weights, numberOfHiddenDegrees = 0)
```

Arguments

varName	name of variable
x	numeric (no NAs/NULLs) effective variable
yC	(no NAs/NULLs) outcome variable
yTarget	scalar target for yC to match (yC==tTarget is goal)
weights	(optional) numeric, non-negative, no NAs/NULLs at least two positive positions
numberOfHiddenDegrees	optional scalar ≥ 0 number of additional modeling degrees of freedom to account for.

Value

significance estimate of a 1-variable logistic regression

Examples

```
d <- data.frame(y=c(1,1,0,0,1,1,0,0,1,1,1,1))
d$x <- seq_len((nrow(d)))
vtreat:::catScore('x',d$x,d$y,1,NULL)
```

designTreatmentsC	<i>Build all treatments for a data frame to predict a categorical outcome.</i>
-------------------	--

Description

Function to design variable treatments for binary prediction of a categorical outcome. Data frame is assumed to have only atomic columns except for dates (which are converted to numeric).

Usage

```
designTreatmentsC(dframe, varlist, outcomename, outcometarget, ...,
  weights = c(), minFraction = 0.02, smFactor = 0, rareCount = 0,
  rareSig = NULL, collarProb = 0, splitFunction = NULL, ncross = 3,
  catScaling = FALSE, verbose = TRUE, parallelCluster = NULL)
```

Arguments

dframe	Data frame to learn treatments from (training data), must have at least 1 row.
varlist	Names of columns to treat (effective variables).
outcomename	Name of column holding outcome variable. <code>dframe[[outcomename]]</code> must be only finite non-missing values.
outcometarget	Value/level of outcome to be considered "success", and there must be a cut such that <code>dframe[[outcomename]]==outcometarget</code> at least twice and <code>dframe[[outcomename]]!=outcometarget</code> at least twice.
...	no additional arguments, declared to forced named binding of later arguments
weights	optional training weights for each row
minFraction	optional minimum frequency a categorical level must have to be converted to an indicator column.
smFactor	optional smoothing factor for impact coding models.
rareCount	optional integer, allow levels with this count or below to be pooled into a shared rare-level. Defaults to 0 or off.
rareSig	optional numeric, suppress levels from pooling at this significance value greater. Defaults to NULL or off.
collarProb	what fraction of the data (pseudo-probability) to collar data at if <code>doCollar</code> is set during prepare .
splitFunction	(optional) see <code>vtreat::buildEvalSets</code> .
ncross	optional scalar ≥ 2 number of cross validation splits use in rescoring complex variables.
catScaling	optional, if TRUE use <code>glm()</code> linkspace, if FALSE use <code>lm()</code> for scaling.
verbose	if TRUE print progress.
parallelCluster	(optional) a cluster object created by package <code>parallel</code> or package <code>snow</code>

Details

The main fields are mostly vectors with names (all with the same names in the same order):

- `vars` : (character array without names) names of variables (in same order as names on the other diagnostic vectors) - `varMoves` : logical TRUE if the variable varied during hold out scoring, only variables that move will be in the treated frame - `#` - `sig` : an estimate significance of effect

See the `vtreat` vignette for a bit more detail and a worked example.

Value

treatment plan (for use with prepare)

See Also

[prepare designTreatmentsN](#)

Examples

```
dTrainC <- data.frame(x=c('a','a','a','b','b','b'),
  z=c(1,2,3,4,5,6),
  y=c(FALSE,FALSE,TRUE,FALSE,TRUE,TRUE))
dTestC <- data.frame(x=c('a','b','c',NA),
  z=c(10,20,30,NA))
treatmentsC <- designTreatmentsC(dTrainC,colnames(dTrainC),'y',TRUE)
dTrainCTreated <- prepare(treatmentsC,dTrainC,pruneSig=0.99)
dTestCTreated <- prepare(treatmentsC,dTestC,pruneSig=0.99)
```

designTreatmentsN *build all treatments for a data frame to predict a numeric outcome*

Description

Function to design variable treatments for binary prediction of a numeric outcome. Data frame is assumed to have only atomic columns except for dates (which are converted to numeric). Note: each column is processed independently of all others.

Usage

```
designTreatmentsN(dframe, varlist, outcomename, ..., weights = c(),
  minFraction = 0.02, smFactor = 0, rareCount = 0, rareSig = NULL,
  collarProb = 0, splitFunction = NULL, ncross = 3, verbose = TRUE,
  parallelCluster = NULL)
```

Arguments

dframe	Data frame to learn treatments from (training data), must have at least 1 row.
varlist	Names of columns to treat (effective variables).
outcomename	Name of column holding outcome variable. <code>dframe[[outcomename]]</code> must be only finite non-missing values and there must be a cut such that <code>dframe[[outcomename]]</code> is both above the cut at least twice and below the cut at least twice.
...	no additional arguments, declared to forced named binding of later arguments
weights	optional training weights for each row
minFraction	optional minimum frequency a categorical level must have to be converted to an indicator column.

smFactor	optional smoothing factor for impact coding models.
rareCount	optional integer, allow levels with this count or below to be pooled into a shared rare-level. Defaults to 0 or off.
rareSig	optional numeric, suppress levels from pooling at this significance value greater. Defaults to NULL or off.
collarProb	what fraction of the data (pseudo-probability) to collar data at if doCollar is set during prepare .
splitFunction	(optional) see vtreat::buildEvalSets .
ncross	optional scalar >=2 number of cross validation splits use in rescoring complex variables.
verbose	if TRUE print progress.
parallelCluster	(optional) a cluster object created by package parallel or package snow

Details

The main fields are mostly vectors with names (all with the same names in the same order):

- vars : (character array without names) names of variables (in same order as names on the other diagnostic vectors) - varMoves : logical TRUE if the variable varied during hold out scoring, only variables that move will be in the treated frame - sig : an estimate significance of effect

See the vtreat vignette for a bit more detail and a worked example.

Value

treatment plan (for use with prepare)

See Also

[prepare designTreatmentsC](#)

Examples

```
dTrainN <- data.frame(x=c('a', 'a', 'a', 'a', 'b', 'b', 'b'),
  z=c(1,2,3,4,5,6,7),y=c(0,0,0,1,0,1,1))
dTestN <- data.frame(x=c('a', 'b', 'c', NA),
  z=c(10,20,30,NA))
treatmentsN = designTreatmentsN(dTrainN,colnames(dTrainN),'y')
dTrainNTreated <- prepare(treatmentsN,dTrainN,pruneSig=0.99)
dTestNTreated <- prepare(treatmentsN,dTestN,pruneSig=0.99)
```

designTreatmentsZ *Design variable treatments with no outcome variable.*

Description

Data frame is assumed to have only atomic columns except for dates (which are converted to numeric). Note: each column is processed independently of all others.

Usage

```
designTreatmentsZ(dframe, varlist, ..., minFraction = 0.02, weights = c(),
  rareCount = 0, collarProb = 0, verbose = TRUE, parallelCluster = NULL)
```

Arguments

dframe	Data frame to learn treatments from (training data), must have at least 1 row.
varlist	Names of columns to treat (effective variables).
...	no additional arguments, declared to forced named binding of later arguments
minFraction	optional minimum frequency a categorical level must have to be converted to an indicator column.
weights	optional training weights for each row
rareCount	optional integer, allow levels with this count or below to be pooled into a shared rare-level. Defaults to 0 or off.
collarProb	what fraction of the data (pseudo-probability) to collar data at if doCollar is set during prepare .
verbose	if TRUE print progress.
parallelCluster	(optional) a cluster object created by package parallel or package snow

Details

The main fields are mostly vectors with names (all with the same names in the same order):

- vars : (character array without names) names of variables (in same order as names on the other diagnostic vectors) - varMoves : logical TRUE if the variable varied during hold out scoring, only variables that move will be in the treated frame

See the vtreat vignette for a bit more detail and a worked example.

Value

treatment plan (for use with [prepare](#))

See Also

[prepare](#) [designTreatmentsC](#) [designTreatmentsN](#)

Examples

```
dTrainZ <- data.frame(x=c('a','a','a','a','b','b',NA,'e','e'),
  z=c(1,2,3,4,5,6,7,NA,9))
dTestZ <- data.frame(x=c('a','x','c',NA),
  z=c(10,20,30,NA))
treatmentsZ = designTreatmentsZ(dTrainZ,colnames(dTrainZ),
  rareCount=0)
dTrainZTreated <- prepare(treatmentsZ,dTrainZ,pruneSig=1)
dTestZTreated <- prepare(treatmentsZ,dTestZ,pruneSig=1)
```

format.vtreatment *Display treatment plan.*

Description

Display treatment plan.

Usage

```
## S3 method for class 'vtreatment'
format(x, ...)
```

Arguments

x	treatment plan
...	additional args (to match general signature).

getSplitPlanAppLabels *read application labels off a split plan.*

Description

read application labels off a split plan.

Usage

```
getSplitPlanAppLabels(nRow, plan)
```

Arguments

nRow	number of rows in original data.frame.
plan	split plan

Value

vector of labels

See Also

[kWayCrossValidation](#), [kWayStratifiedY](#), and [makekWayCrossValidationGroupedByColumn](#)

Examples

```
plan <- oneWayHoldout(3, NULL, NULL, NULL)
getSplitPlanAppLabels(3, plan)
```

`kWayCrossValidation` *k-fold cross validation, a splitFunction in the sense of vtreat::buildEvalSets*

Description

k-fold cross validation, a `splitFunction` in the sense of `vtreat::buildEvalSets`

Usage

```
kWayCrossValidation(nRows, nSplits, dframe, y)
```

Arguments

<code>nRows</code>	number of rows to split (>1).
<code>nSplits</code>	number of groups to split into (>1, <=nRows).
<code>dframe</code>	original data frame (ignored).
<code>y</code>	numeric outcome variable (ignored).

Value

split plan

Examples

```
kWayCrossValidation(7, 2, NULL, NULL)
```

kWayStratifiedY	<i>k-fold cross validation stratified on y, a splitFunction in the sense of vtreat::buildEvalSets</i>
-----------------	---

Description

k-fold cross validation stratified on y, a splitFunction in the sense of vtreat::buildEvalSets

Usage

```
kWayStratifiedY(nRows, nSplits, dframe, y)
```

Arguments

nRows	number of rows to split (>1)
nSplits	number of groups to split into (<nRows,>1).
dframe	original data frame (ignored).
y	numeric outcome variable try to have equidistributed in each split.

Value

split plan

Examples

```
set.seed(23255)
d <- data.frame(y=sin(1:100))
pStrat <- kWayStratifiedY(nrow(d),5,d,d$y)
problemAppPlan(nrow(d),5,pStrat,TRUE)
d$stratGroup <- vtreat::getSplitPlanAppLabels(nrow(d),pStrat)
pSimple <- kWayCrossValidation(nrow(d),5,d,d$y)
problemAppPlan(nrow(d),5,pSimple,TRUE)
d$simpleGroup <- vtreat::getSplitPlanAppLabels(nrow(d),pSimple)
summary(tapply(d$y,d$simpleGroup,mean))
# ggplot(data=d,aes(x=y,color=as.factor(simpleGroup))) +
#   geom_density() + ggtitle('simple grouping')
summary(tapply(d$y,d$stratGroup,mean))
# ggplot(data=d,aes(x=y,color=as.factor(stratGroup))) +
#   geom_density() + ggtitle('y-stratified grouping')

# # And you can (and should) use your own functions or libraries.
# splitFn <- function(nRows,nSplits,dframe,y) {
#   fullSeq <- seq_len(nRows)
#   part <- caret::createFolds(y=y,k=nSplits)
#   lapply(part,
#     function(appi) {
```

```

#           list(train=setdiff(fullSeq, appi), app=appi)
#           })
# }
# pCaret <- splitFn(nrow(d), 5, d, d$y)
# problemAppPlan(nrow(d), 5, pCaret, TRUE)
# d$caretGroup <- vtreat::getSplitPlanAppLabels(nrow(d), pCaret)
# ggplot(data=d, aes(x=y, color=as.factor(caretGroup))) +
#   geom_density() + ggtitle('caret::createFolds grouping')

```

linScore *Return in-sample linear stats and scaling.*

Description

Return in-sample linear stats and scaling.

Usage

```
linScore(varName, xcol, ycol, weights, numberOfHiddenDegrees = 0)
```

Arguments

varName	name of variable
xcol	numeric vector of inputs (no NA/NULL/NaN)
ycol	numeric vector of outcomes (no NA/NULL/NaN)
weights	numeric vector of data weights (no NA/NULL/NaN, all>0.0)
numberOfHiddenDegrees	optional scalar ≥ 0 number of additional modeling degrees of freedom to account for.

Value

significance estimate and scaling.

makekWayCrossValidationGroupedByColumn
Build a k-fold cross validation splitter, respecting (never splitting) groupingColumn.

Description

Build a k-fold cross validation splitter, respecting (never splitting) groupingColumn.

Usage

```
makekWayCrossValidationGroupedByColumn(groupingColumnName)
```

Arguments

```
groupingColumnName
    name of column to group by.
```

Value

splitting function in the sense of vtreat::buildEvalSets.

Examples

```
d <- data.frame(y=sin(1:100))
d$group <- floor(seq_len(nrow(d))/5)
splitter <- makekWayCrossValidationGroupedByColumn('group')
split <- splitter(nrow(d),5,d,d$y)
d$splitLabel <- vtreat::getSplitPlanAppLabels(nrow(d),split)
rowSums(table(d$group,d$splitLabel)>0)
```

```
mkCrossFrameCExperiment
```

Run categorical cross-frame experiment.

Description

Builds a [designTreatmentsC](#) treatment plan and a data frame prepared from dframe that is "cross" in the sense each row is treated using a treatment plan built from a subset of dframe disjoint from the given row. The goal is to try to and supply a method of breaking nested model bias other than splitting into calibration, training, test sets.

Usage

```
mkCrossFrameCExperiment(dframe, varlist, outcomename, outcometarget, ...,
  weights = c(), minFraction = 0.02, smFactor = 0, rareCount = 0,
  rareSig = 1, collarProb = 0, scale = FALSE, doCollar = FALSE,
  splitFunction = NULL, ncross = 3, catScaling = FALSE,
  parallelCluster = NULL)
```

Arguments

```
dframe          Data frame to learn treatments from (training data), must have at least 1 row.
varlist         Names of columns to treat (effective variables).
outcomename     Name of column holding outcome variable. dframe[[outcomename]] must be
                only finite non-missing values.
```

outcometarget	Value/level of outcome to be considered "success", and there must be a cut such that <code>dframe[[outcomename]]==outcometarget</code> at least twice and <code>dframe[[outcomename]]!=outcometarget</code> at least twice.
...	no additional arguments, declared to forced named binding of later arguments
weights	optional training weights for each row
minFraction	optional minimum frequency a categorical level must have to be converted to an indicator column.
smFactor	optional smoothing factor for impact coding models.
rareCount	optional integer, allow levels with this count or below to be pooled into a shared rare-level. Defaults to 0 or off.
rareSig	optional numeric, suppress levels from pooling at this significance value greater. Defaults to NULL or off.
collarProb	what fraction of the data (pseudo-probability) to collar data at if <code>doCollar</code> is set during prepare .
scale	optional if TRUE replace numeric variables with regression ("move to outcome-scale").
doCollar	optional if TRUE collar numeric variables by cutting off after a tail-probability specified by <code>collarProb</code> during treatment design.
splitFunction	(optional) see <code>vtreat::buildEvalSets</code> .
ncross	optional scalar ≥ 2 number of cross-validation rounds to design.
catScaling	optional, if TRUE use <code>glm()</code> linkspace, if FALSE use <code>lm()</code> for scaling.
parallelCluster	(optional) a cluster object created by package <code>parallel</code> or package <code>snow</code>

Value

list with treatments and crossFrame

See Also

[designTreatmentsC](#) [designTreatmentsN](#) [prepare](#)

Examples

```
set.seed(23525)
zip <- paste('z',1:100)
N = 200
d <- data.frame(zip=sample(zip,N,replace=TRUE),
                zip2=sample(zip,20,replace=TRUE),
                y=runif(N))
del <- runif(length(zip))
names(del) <- zip
d$y <- d$y + del[d$zip2]
d$yc <- d$y>=mean(d$y)
cC <- mkCrossFrameCExperiment(d,c('zip', 'zip2'), 'yc', TRUE,
```



```

rareCount=2,rareSig=0.9)
cor(as.numeric(cC$crossFrame$yc),cC$crossFrame$zip_catB) # poor
cor(as.numeric(cC$crossFrame$yc),cC$crossFrame$zip2_catB) # better
treatments <- cC$treatments
dTrainV <- cC$crossFrame

```

mkCrossFrameNExperiment

Run numeric cross frame experiment.

Description

Builds a [designTreatmentsC](#) treatment plan and a data frame prepared from `dframe` that is "cross" in the sense each row is treated using a treatment plan built from a subset of `dframe` disjoint from the given row. The goal is to try to and supply a method of breaking nested model bias other than splitting into calibration, training, test sets.

Usage

```

mkCrossFrameNExperiment(dframe, varlist, outcomename, ..., weights = c(),
  minFraction = 0.02, smFactor = 0, rareCount = 0, rareSig = 1,
  collarProb = 0, scale = FALSE, doCollar = FALSE, splitFunction = NULL,
  ncross = 3, parallelCluster = NULL)

```

Arguments

<code>dframe</code>	Data frame to learn treatments from (training data), must have at least 1 row.
<code>varlist</code>	Names of columns to treat (effective variables).
<code>outcomename</code>	Name of column holding outcome variable. <code>dframe[[outcomename]]</code> must be only finite non-missing values and there must be a cut such that <code>dframe[[outcomename]]</code> is both above the cut at least twice and below the cut at least twice.
<code>...</code>	no additional arguments, declared to forced named binding of later arguments
<code>weights</code>	optional training weights for each row
<code>minFraction</code>	optional minimum frequency a categorical level must have to be converted to an indicator column.
<code>smFactor</code>	optional smoothing factor for impact coding models.
<code>rareCount</code>	optional integer, allow levels with this count or below to be pooled into a shared rare-level. Defaults to 0 or off.
<code>rareSig</code>	optional numeric, suppress levels from pooling at this significance value greater. Defaults to NULL or off.
<code>collarProb</code>	what fraction of the data (pseudo-probability) to collar data at if <code>doCollar</code> is set during prepare .
<code>scale</code>	optional if TRUE replace numeric variables with regression ("move to outcome-scale").

doCollar optional if TRUE collar numeric variables by cutting off after a tail-probability specified by collarProb during treatment design.

splitFunction (optional) see vtreat::buildEvalSets .

ncross optional scalar ≥ 2 number of cross-validation rounds to design.

parallelCluster (optional) a cluster object created by package parallel or package snow

Value

treatment plan (for use with prepare)

See Also

[designTreatmentsC](#) [designTreatmentsN](#) [prepare](#)

Examples

```
set.seed(23525)
zip <- paste('z',1:100)
N = 200
d <- data.frame(zip=sample(zip,N,replace=TRUE),
                zip2=sample(zip,N,replace=TRUE),
                y=runif(N))
del <- runif(length(zip))
names(del) <- zip
d$y <- d$y + del[d$zip2]
d$yc <- d$y >= mean(d$y)
cN <- mkCrossFrameNExperiment(d,c('zip', 'zip2'), 'y',
                             rareCount=2, rareSig=0.9)
cor(cN$crossFrame$y, cN$crossFrame$zip_catN) # poor
cor(cN$crossFrame$y, cN$crossFrame$zip2_catN) # better
treatments <- cN$treatments
dTrainV <- cN$crossFrame
```

oneWayHoldout	<i>One way holdout, a splitFunction in the sense of vtreat::buildEvalSets. Doesn't respect nSplits.</i>
---------------	---

Description

One way holdout, a splitFunction in the sense of vtreat::buildEvalSets. Doesn't respect nSplits.

Usage

```
oneWayHoldout(nRows, nSplits, dframe, y)
```

Arguments

nRows	number of rows to split (integer >1).
nSplits	number of groups to split into (ignored).
dframe	original data frame (ignored).
y	numeric outcome variable (ignored).

Value

split plan

Examples

```
oneWayHoldout(3, NULL, NULL, NULL)
```

prepare	<i>Apply treatments and restrict to useful variables.</i>
---------	---

Description

Use a treatment plan to prepare a data frame for analysis. The resulting frame will have new effective variables that are numeric and free of NaN/NA. If the outcome column is present it will be copied over. The intent is that these frames are compatible with more machine learning techniques, and avoid a lot of corner cases (NA,NaN, novel levels, too many levels). Note: each column is processed independently of all others. Also copies over outcome if present.

Usage

```
prepare(treatmentplan, dframe, pruneSig, ..., scale = FALSE,
        doCollar = FALSE, varRestriction = c(), parallelCluster = NULL)
```

Arguments

treatmentplan	Plan built by designTreatmentsC() or designTreatmentsN()
dframe	Data frame to be treated
pruneSig	suppress variables with significance above this level
...	no additional arguments, declared to forced named binding of later arguments
scale	optional if TRUE replace numeric variables with single variable model regressions ("move to outcome-scale"). These have mean zero and (for variables with significant less than 1) slope 1 when regressed (lm for regression problems/glm for classification problems) against outcome.
doCollar	optional if TRUE collar numeric variables by cutting off after a tail-probability specified by collarProb during treatment design.
varRestriction	optional list of treated variable names to restrict to
parallelCluster	(optional) a cluster object created by package parallel or package snow

Value

treated data frame (all columns numeric, without NA,NaN)

See Also

[designTreatmentsC](#) [designTreatmentsN](#)

Examples

```
dTrainN <- data.frame(x=c('a','a','a','a','b','b','b'),
  z=c(1,2,3,4,5,6,7),y=c(0,0,0,1,0,1,1))
dTestN <- data.frame(x=c('a','b','c',NA),z=c(10,20,30,NA))
treatmentsN = designTreatmentsN(dTrainN,colnames(dTrainN),'y')
dTrainNTreated <- prepare(treatmentsN,dTrainN,1.0)
dTestNTreated <- prepare(treatmentsN,dTestN,1.0)

dTrainC <- data.frame(x=c('a','a','a','b','b','b'),
  z=c(1,2,3,4,5,6),y=c(FALSE,FALSE,TRUE,FALSE,TRUE,TRUE))
dTestC <- data.frame(x=c('a','b','c',NA),z=c(10,20,30,NA))
treatmentsC <- designTreatmentsC(dTrainC,colnames(dTrainC),'y',TRUE)
dTrainCTreated <- prepare(treatmentsC,dTrainC,1.0)
dTestCTreated <- prepare(treatmentsC,dTestC,1.0)
```

print.vtreatment *Print treatmentplan.*

Description

Print treatmentplan.

Usage

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

Arguments

x	treatmentplan
...	additional args (to match general signature).

See Also

[designTreatmentsC](#) [designTreatmentsN](#) [prepare](#)

problemAppPlan	<i>check if appPlan is a good carve-up of 1:nRows into nSplits groups</i>
----------------	---

Description

check if appPlan is a good carve-up of 1:nRows into nSplits groups

Usage

```
problemAppPlan(nRows, nSplits, appPlan, strictCheck)
```

Arguments

nRows	number of rows to carve-up
nSplits	number of sets to carve-up into
appPlan	carve-up to critique
strictCheck	logical, if true expect application data to be a carve-up and training data to be a maximal partition and to match nSplits.

Value

problem with carve-up (null if good)

See Also

[kWayCrossValidation](#), [kWayStratifiedY](#), and [makekWayCrossValidationGroupedByColumn](#)

Examples

```
plan <- oneWayHoldout(3, NULL, NULL, NULL)
problemAppPlan(3, 3, plan, TRUE)
```

vnames	<i>New treated variable names from a treatmentplan\$treatment item.</i>
--------	---

Description

New treated variable names from a treatmentplan\$treatment item.

Usage

```
vnames(x)
```

Arguments

x vtreatment item

See Also

[designTreatmentsC](#) [designTreatmentsN](#)

vorig

Original variable name from a treatmentplan\$treatment item.

Description

Original variable name from a treatmentplan\$treatment item.

Usage

vorig(x)

Arguments

x vtreatment item.

See Also

[designTreatmentsC](#) [designTreatmentsN](#)

Index

*Topic **package**

- vtreat-package, 2
- buildEvalSets, 4
- catScore, 6
- designTreatmentsC, 2, 6, 9, 10, 15–18, 20, 22
- designTreatmentsN, 2, 8, 8, 10, 16, 18, 20, 22
- designTreatmentsZ, 10
- format.vtreatment, 11
- getSplitPlanAppLabels, 11
- kWayCrossValidation, 5, 12, 12, 21
- kWayStratifiedY, 5, 12, 13, 21
- linScore, 14
- makekWayCrossValidationGroupedByColumn,
5, 12, 14, 21
- mkCrossFrameCExperiment, 15
- mkCrossFrameNExperiment, 17
- oneWayHoldout, 18
- prepare, 2, 7–10, 16–18, 19, 20
- print.vtreatment, 20
- problemAppPlan, 21
- vnames, 21
- vorig, 22
- vtreat (vtreat-package), 2
- vtreat-package, 2