Package ‘vtreat’

December 8, 2019

Type Package
Title A Statistically Sound 'data.frame' Processor/Conditioner
Version 1.4.8
Date 2019-12-08
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. 'vtreat' 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, and new categorical levels (levels seen during application, but not during training). Reference: "'vtreat': a data.frame Processor for Predictive Modeling", Zumel, Mount, 2016, <DOI:10.5281/zenodo.1173313>.
License GPL-2 | GPL-3
Depends R (>= 3.2.1)
Imports stats, parallel, wrapr (>= 1.9.3)
Suggests rquery (>= 1.3.9), rqdatatable (>= 1.2.3), isotone, lme4, knitr, rmarkdown, data.table (>= 1.11.4), DBI, RSQLite, datasets, R.rsp, RUnit
LazyData true
VignetteBuilder knitr, R.rsp
RoxygenNote 7.0.2
ByteCompile true
NeedsCompilation no
Author John Mount [aut, cre], Nina Zumel [aut], Win-Vector LLC [cph]
Repository: CRAN
Date/Publication: 2019-12-08 19:00:03 UTC

R topics documented:

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>as_rquery_plan</td>
<td>3</td>
</tr>
<tr>
<td>BinomialOutcomeTreatment</td>
<td>4</td>
</tr>
<tr>
<td>buildEvalSets</td>
<td>5</td>
</tr>
<tr>
<td>center_scale</td>
<td>7</td>
</tr>
<tr>
<td>classification_parameters</td>
<td>8</td>
</tr>
<tr>
<td>designTreatmentsC</td>
<td>9</td>
</tr>
<tr>
<td>designTreatmentsN</td>
<td>11</td>
</tr>
<tr>
<td>designTreatmentsZ</td>
<td>13</td>
</tr>
<tr>
<td>design_missingness_treatment</td>
<td>15</td>
</tr>
<tr>
<td>format.vtreatment</td>
<td>16</td>
</tr>
<tr>
<td>getSplitPlanAppLabels</td>
<td>17</td>
</tr>
<tr>
<td>kWayCrossValidation</td>
<td>17</td>
</tr>
<tr>
<td>kWayStratifiedY</td>
<td>18</td>
</tr>
<tr>
<td>kWayStratifiedYReplace</td>
<td>19</td>
</tr>
<tr>
<td>makekWayCrossValidationGroupedByColumn</td>
<td>20</td>
</tr>
<tr>
<td>mkCrossFrameCEExperiment</td>
<td>20</td>
</tr>
<tr>
<td>mkCrossFrameMExperiment</td>
<td>23</td>
</tr>
<tr>
<td>mkCrossFrameNExperiment</td>
<td>25</td>
</tr>
<tr>
<td>MultinomialOutcomeTreatment</td>
<td>27</td>
</tr>
<tr>
<td>multinomial_parameters</td>
<td>28</td>
</tr>
<tr>
<td>novel_value_summary</td>
<td>29</td>
</tr>
<tr>
<td>NumericOutcomeTreatment</td>
<td>30</td>
</tr>
<tr>
<td>oneWayHoldout</td>
<td>31</td>
</tr>
<tr>
<td>patch_columns_into_frame</td>
<td>31</td>
</tr>
<tr>
<td>ppCoderC</td>
<td>32</td>
</tr>
<tr>
<td>ppCoderN</td>
<td>33</td>
</tr>
<tr>
<td>prepare</td>
<td>33</td>
</tr>
<tr>
<td>prepare.multinomial_plan</td>
<td>34</td>
</tr>
<tr>
<td>prepare.simple_plan</td>
<td>35</td>
</tr>
<tr>
<td>prepare.treatmentplan</td>
<td>36</td>
</tr>
<tr>
<td>pre_comp_xval</td>
<td>38</td>
</tr>
<tr>
<td>print.multinomial_plan</td>
<td>38</td>
</tr>
<tr>
<td>print.simple_plan</td>
<td>39</td>
</tr>
<tr>
<td>print.treatmentplan</td>
<td>39</td>
</tr>
<tr>
<td>print.vtreatment</td>
<td>40</td>
</tr>
<tr>
<td>problemAppPlan</td>
<td>40</td>
</tr>
<tr>
<td>regression_parameters</td>
<td>41</td>
</tr>
<tr>
<td>rquery_prepare</td>
<td>41</td>
</tr>
<tr>
<td>run_vtreat_tests</td>
<td>43</td>
</tr>
<tr>
<td>solvesIsotone</td>
<td>44</td>
</tr>
<tr>
<td>solveNonDecreasing</td>
<td>45</td>
</tr>
<tr>
<td>solveNonIncreasing</td>
<td>46</td>
</tr>
</tbody>
</table>
### as_rquery_plan

**Convert vtreatment plans into a sequence of rquery operations.**

**Description**

Convert vtreatment plans into a sequence of rquery operations.

**Usage**

```r
as_rquery_plan(treatmentplans, ..., var_restriction = NULL)
```

**Arguments**

- `treatmentplans` vtreat treatment plan or list of vtreat treatment plan sharing same outcome and outcome type.
- `...` not used, force any later arguments to bind to names.
- `var_restriction` character, if not null restrict to producing these variables.

**Value**

list(optree_generator (ordered list of functions), temp_tables (named list of tables))

**See Also**

rquery_prepare
BinomialOutcomeTreatment

Stateful object for designing and applying binomial outcome treatments.

Description

Hold settings are results for binomial classification data preparation.

Usage

BinomialOutcomeTreatment(
    ..., var_list,
buildEvalSets

```
outcome_name,
outcome_target,
cols_to_copy = NULL,
params = NULL,
imputation_map = NULL
)
```

Arguments

... not used, force arguments to be specified by name.
var_list Names of columns to treat (effective variables).
outcome_name Name of column holding outcome variable. `dframe[[outcome_name]]` must be only finite non-missing values.
outcome_target Value/level of outcome to be considered "success", and there must be a cut such that `dframe[[outcome_name]]==outcome_target` at least twice and `dframe[[outcome_name]]!=outcome_target` at least twice.
cols_to_copy list of extra columns to copy.
params parameters list from `classification_parameters`
imputation_map map from column names to functions of signature f(values: numeric, weights: numeric), simple missing value imputers.

Details

Please see `https://github.com/WinVector/vtreat/blob/master/Examples/fit_transform/fit_transform_api.md, mkCrossFrameCExperiment, designTreatmentsC`, and `prepare.treatmentplan` for details.

```
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

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

Details

Also sets attribute "splitmethod" on return value that describes how the split was performed. attr(returnValue,'splitmethod') is one of: 'nonsplit' (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

```r
# use
buildEvalSets(200)

# longer example
# helper fns
# fit models using experiment plan to estimate out of sample behavior
fitModelAndApply <- function(trainData, applicationData) {
  model <- lm(y~x, data=trainData)
```
predict(model,newdata=applicationData)
}
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)

---

**center_scale**

*Center and scale a set of variables.*

**Description**

Center and scale a set of variables. Other columns are passed through.

**Usage**

```
center_scale(d, center, scale)
```

**Arguments**

- `d` : data.frame to work with
- `center` : named vector of variables to center
- `scale` : named vector of variables to scale
Value

d with centered and scaled columns altered

Examples

d <- data.frame(x = 1:5,
y = c('a', 'a', 'b', 'b', 'b'))
vars_to_transform = "x"
t <- base::scale(as.matrix(d[, vars_to_transform, drop = FALSE]),
                 center = TRUE, scale = TRUE)
t
centering <- attr(t, "scaled:center")
scaling <- attr(t, "scaled:scale")
center_scale(d, center = centering, scale = scaling)

---

classification_parameters

vtreat classification parameters.

Description

A list of settings and values for vtreat binomial classification fitting. Please see https://github.com/WinVector/vtreat/blob/master/Examples/fit_transform/fit_transform_api.md, mkCrossFrameCExperiment, designTreatmentsC, and prepare.treatmentplan for details.

Usage

classification_parameters(user_params = NULL)

Arguments

user_params list of user overrides.

Value

filled out parameter list
designTreatmentsC

Build all treatments for a data frame to predict a categorical outcome.

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). Note: re-encoding high cardinality categorical variables can introduce undesirable nested model bias, for such data consider using `mkCrossFrameCExperiment`.

Usage

```r
designTreatmentsC(
  dframe,
  varlist,
  outcomename,
  outcometarget,
  ..., 
  weights = c(),
  minFraction = 0.02,
  smFactor = 0,
  rareCount = 0,
  rareSig = NULL,
  collarProb = 0,
  codeRestriction = NULL,
  customCoders = NULL,
  splitFunction = NULL,
  ncross = 3,
  forceSplit = FALSE,
  catScaling = TRUE,
  verbose = TRUE,
  parallelCluster = NULL,
  use_parallel = TRUE,
  missingness_imputation = NULL,
  imputation_map = 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 `dframe[[outcomename]] == outcometarget` at least twice and `dframe[[outcomename]] != outcometarget` 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.treatmentplan.
codeRestriction what types of variables to produce (character array of level codes, NULL means no restriction).
customCoders map from code names to custom categorical variable encoding functions (please see https://github.com/WinVector/vtreat/blob/master/extras/CustomLevelCoders.md).
splitFunction (optional) see vtreat::buildEvalSets.
ncross optional scalar >=2 number of cross validation splits use in rescoring complex variables.
forceSplit logical, if TRUE force cross-validated significance calculations on all variables.
catScaling optional, if TRUE use glm() linkspace, if FALSE use lm() for scaling.
verbose if TRUE print progress.
parallelCluster (optional) a cluster object created by package parallel or package snow.
use_parallel logical, if TRUE use parallel methods (when parallel cluster is set).
missingness_imputation function of signature f(values: numeric, weights: numeric), simple missing value imputer.
imputation_map map from column names to functions of signature f(values: numeric, weights: numeric), simple missing value imputers.

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.
Columns that do not vary are not passed through.

Value
treatment plan (for use with prepare)
designTreatmentsN

See Also

prepare.treatmentplan, designTreatmentsN, designTreatmentsZ, mkCrossFrameCExperiment

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. Note: re-encoding high cardinality categorical variables can introduce undesirable nested model bias, for such data consider using mkCrossFrameNExperiment.

Usage

designTreatmentsN(
  dframe,
  varlist,
  outcomeName,
  ...
  weights = c(),
  minFraction = 0.02,
  smFactor = 0,
  rareCount = 0,
  rareSig = NULL,
  collarProb = 0,
  codeRestriction = NULL,
  customCoders = NULL,
  splitFunction = NULL,
  ncross = 3,
  forceSplit = FALSE,
  verbose = TRUE,
  parallelCluster = NULL,
  use_parallel = TRUE,
missingness_imputation = NULL,
imputation_map = 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).
outcome Name of column holding outcome variable. dframe[[outcome]] must be
only finite non-missing values and there must be a cut such that dframe[[outcome]]
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.treatmentplan.
codeRestriction what types of variables to produce (character array of level codes, NULL means
no restriction).
customCoders map from code names to custom categorical variable encoding functions (please
md).
splitFunction (optional) see vtreat::buildEvalSets.
ncross optional scalar >=2 number of cross validation splits use in rescoring complex
variables.
forceSplit logical, if TRUE force cross-validated significance calculations on all variables.
verbose if TRUE print progress.
parallelCluster (optional) a cluster object created by package parallel or package snow.
use_parallel logical, if TRUE use parallel methods (when parallel cluster is set).
missingness_imputation function of signature f(values: numeric, weights: numeric), simple missing
value imputer.
imputation_map map from column names to functions of signature f(values: numeric, weights:
numeric), simple missing value imputers.
**designTreatmentsZ**

**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.

Columns that do not vary are not passed through.

**Value**

treatment plan (for use with prepare)

**See Also**

`prepare.treatmentplan, designTreatmentsC, designTreatmentsZ, mkCrossFrameNExperiment`

**Examples**

```r
dTrainN <- data.frame(x=c('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,
  weights = c(),
  rareCount = 0,
  collarProb = 0,
  codeRestriction = NULL,
)
customCoders = NULL,
verbose = TRUE,
parallelCluster = NULL,
use_parallel = TRUE,
missingness_imputation = NULL,
imputation_map = 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.treatmentplan.

codeRestriction what types of variables to produce (character array of level codes, NULL means no restriction).
customCoders map from code names to custom categorical variable encoding functions (please see https://github.com/WinVector/vtreat/blob/master/extras/CustomLevelCoders.md).
verbose if TRUE print progress.
parallelCluster (optional) a cluster object created by package parallel or package snow.
use_parallel logical, if TRUE use parallel methods (if parallel cluster is set).
missingness_imputation function of signature f(values: numeric, weights: numeric), simple missing value imputer.
imputation_map map from column names to functions of signature f(values: numeric, weights: numeric), simple missing value imputers.

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.
Columns that do not vary are not passed through.
**design_missingness_treatment**

Design a simple treatment plan to indicate missingness and perform simple imputation.

### Description

Design a simple treatment plan to indicate missingness and perform simple imputation.

### Usage

```r
design_missingness_treatment(
    dframe,
    ..., 
    varlist = colnames(dframe),
    invalid_mark = "_invalid_",
    drop_constant_columns = FALSE
)
```

### Arguments

- **dframe**
  - data.frame to drive design.
- **...**
  - not used, forces later arguments to bind by name.
- **varlist**
  - character, names of columns to process.
- **invalid_mark**
  - character, name to use for NA levels and novel levels.
- **drop_constant_columns**
  - logical, if TRUE drop columns that do not vary from the treatment plan.
Value

simple treatment plan.

See Also

prepare.simple_plan

Examples

d <- wrapr::build_frame(
  "x1", "x2", "x3" |
  1 , 4 , "A" |
  NA , 5 , "B" |
  3 , 6 , NA )

plan <- design_missingness_treatment(d)
prepare(plan, d)

prepare(plan, data.frame(x1=NA, x2=NA, x3="E"))

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 <- kWayStratifiedY(3,2,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**

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

**Value**

split plan

**Examples**

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

---

**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**

```r
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))
summary(tapply(d$y,d$stratGroup,mean))
```

---

**kWayStratifiedYReplace**

_k-fold cross validation stratified with replacement on\ y, a splitFunction in the sense of vtreat::buildEvalSets_.

---

**Description**

Build a \(k\)-fold cross validation sample where training sets are the same size as the original data, and built by sampling disjoint from test/application sets (sampled with replacement).

**Usage**

```r
kWayStratifiedYReplace(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**

```r
set.seed(23255)
d <- data.frame(y=sin(1:100))
pStrat <- kWayStratifiedYReplace(nrow(d),5,d,d$y)
```
**Description**

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

**Usage**

```r
makekWayCrossValidationGroupedByColumn(groupingColumnName)
```

**Arguments**

- `groupingColumnName`
  
  name of column to group by.

**Value**

splitting function in the sense of vtreat::buildEvalSets.

**Examples**

```r
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)
```

---

**Description**

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

\[ \text{mkCrossFrameCExperiment}( \]
\[ \text{dframe,} \]
\[ \text{varlist,} \]
\[ \text{outcomename,} \]
\[ \text{outcometarget,} \]
\[ ... \]
\[ \text{weights = c()}, \]
\[ \text{minFraction = 0.02,} \]
\[ \text{smFactor = 0,} \]
\[ \text{rareCount = 0,} \]
\[ \text{rareSig = 1,} \]
\[ \text{collarProb = 0,} \]
\[ \text{codeRestriction = NULL,} \]
\[ \text{customCoders = NULL,} \]
\[ \text{scale = FALSE,} \]
\[ \text{doCollar = FALSE,} \]
\[ \text{splitFunction = NULL,} \]
\[ \text{ncross = 3,} \]
\[ \text{forceSplit = FALSE,} \]
\[ \text{catScaling = TRUE,} \]
\[ \text{verbose = TRUE,} \]
\[ \text{parallelCluster = NULL,} \]
\[ \text{use_parallel = TRUE,} \]
\[ \text{missingness_imputation = NULL,} \]
\[ \text{imputation_map = NULL} \]
\)

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 dframe[[outcomename]]==outcometarget at least twice and dframe[[outcomename]]!=outcometarget at least twice.
...
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.treatmentplan.
codeRestriction what types of variables to produce (character array of level codes, NULL means no restriction).
customCoders map from code names to custom categorical variable encoding functions (please see https://github.com/WinVector/vtreat/blob/master/extras/CustomLevelCoders.md).
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 collarProb during treatment design.
splitFunction (optional) see vtreart::buildEvalSets.
ncross optional scalar >= 2 number of cross-validation rounds to design.
forceSplit logical, if TRUE force cross-validated significance calculations on all variables.
catScaling optional, if TRUE use glm() linkspace, if FALSE use lm() for scaling.
verbose if TRUE print progress.
parallelCluster (optional) a cluster object created by package parallel or package snow.
use_parallel logical, if TRUE use parallel methods.
missingness_imputation function of signature f(values: numeric, weights: numeric), simple missing value imputer.
imputation_map map from column names to functions of signature f(values: numeric, weights: numeric), simple missing value imputers.

Value
list with treatments and crossFrame

See Also
designTreatmentsC, designTreatmentsN, prepare.treatmentplan

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$zip2]
\begin{verbatim}
d$yc <- d$y>=mean(d$y)
cC <- mkCrossFrameCExperiment(d,c('zip1','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
\end{verbatim}

\begin{verbatim}
mkCrossFrameMExperiment
  Function to build multi-outcome vtreat cross frame and treatment plan.

Description

Please see vignette("MultiClassVtreat",package = "vtreat") https://winvector.github.io/vtreat/articles/MultiClassVtreat.html.

Usage

mkCrossFrameMExperiment(
  d,
  vars,
  y_name,
  ..., 
  weights = c(),
  minFraction = 0.02,
  smFactor = 0,
  rareCount = 0,
  rareSig = 1,
  collarProb = 0,
  codeRestriction = NULL,
  customCoders = NULL,
  scale = FALSE,
  doCollar = FALSE,
  splitFunction = NULL,
  ncross = 3,
  forceSplit = FALSE,
  catScaling = FALSE,
  y_dependent_treatments = c("catB"),
  verbose = FALSE,
  parallelCluster = NULL,
  use_parallel = TRUE,
  missingness_imputation = NULL,
  imputation_map = NULL
)
\end{verbatim}
Arguments

d d data to learn from
vars character, vector of independent variable column names.
y_name character, name of outcome column.
... not used, 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.multinomial_plan.
codeRestriction what types of variables to produce (character array of level codes, NULL means no restriction).
customCoders map from code names to custom categorical variable encoding functions (please see https://github.com/WinVector/vtreat/blob/master/extras/CustomLevelCoders.md).
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 collarProb during treatment design.
splitFunction (optional) see vtreat::buildEvalSets.
ncross optional scalar>=2 number of cross-validation rounds to design.
forceSplit logical, if TRUE force cross-validated significance calculations on all variables.
catScaling optional, if TRUE use glm() linkspace, if FALSE use lm() for scaling.
y_dependent_treatments character what treatment types to build per-outcome level.
verbose if TRUE print progress.
parallelCluster (optional) a cluster object created by package parallel or package snow.
use_parallel logical, if TRUE use parallel methods.
missingness_imputation function of signature f(values: numeric, weights: numeric), simple missing value imputer.
imputation_map map from column names to functions of signature f(values: numeric, weights: numeric), simple missing value imputers.
Run a numeric cross frame experiment.

Builds a designTreatmentsN 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

```r
mkCrossFrameNExperiment(
  dframe,
  varlist,
  outcomename,
  ..., 
  weights = c(),
  minFraction = 0.02,
  smFactor = 0,
  rareCount = 0,
  rareSig = 1,
  collarProb = 0,
  codeRestriction = NULL,
  customCoders = NULL,
  scale = FALSE,
  doCollar = FALSE,
  splitFunction = NULL,
  ncross = 3,
  forceSplit = FALSE,
  verbose = TRUE,
  parallelCluster = NULL,
  use_parallel = TRUE,
  missingness_imputation = NULL,
  imputation_map = NULL
)
```
Arguments

- **dfname**: Data frame to learn treatments from (training data), must have at least 1 row.
- **varlist**: Names of columns to treat (effective variables).
- **outcome**: Name of column holding outcome variable. `dframe[[outcome]]` must be only finite non-missing values and there must be a cut such that `dframe[[outcome]]` is both above the cut at least twice and below the cut at least twice.
- **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.treatmentplan`.
- **codeRestriction**: what types of variables to produce (character array of level codes, NULL means no restriction).
- **customCoders**: map from code names to custom categorical variable encoding functions (please see [https://github.com/WinVector/vtreat/blob/master/extras/CustomLevelCoders.md](https://github.com/WinVector/vtreat/blob/master/extras/CustomLevelCoders.md)).
- **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 collarProb during treatment design.
- **splitFunction** (optional) see `vtreat::buildEvalSets`.
- **ncross**: optional scalar>=2 number of cross-validation rounds to design.
- **forceSplit**: logical, if TRUE force cross-validated significance calculations on all variables.
- **verbose**: if TRUE print progress.
- **parallelCluster**: (optional) a cluster object created by package parallel or package snow.
- **use_parallel**: logical, if TRUE use parallel methods.
- **missingness_imputation**: function of signature f(values: numeric, weights: numeric), simple missing value imputer.
- **imputation_map**: map from column names to functions of signature f(values: numeric, weights: numeric), simple missing value imputers.

Value

treatment plan (for use with `prepare`)
MultinomialOutcomeTreatment

Stateful object for designing and applying multinomial outcome treatments.

Description
Hold settings are results for multinomial classification data preparation.

Usage
MultinomialOutcomeTreatment(
  ..., var_list,
  outcome_name,
  cols_to_copy = NULL,
  params = NULL,
  imputation_map = NULL
)

Arguments
... not used, force arguments to be specified by name.
var_list Names of columns to treat (effective variables).

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
multinomial_parameters

outcome_name: Name of column holding outcome variable. `dframe[[outcome_name]]` must be only finite non-missing values.

cols_to_copy: list of extra columns to copy.

params: parameters list from `multinomial_parameters`

imputation_map: map from column names to functions of signature `f(values: numeric, weights: numeric)`, simple missing value imputers.

Details

Please see `https://github.com/WinVector/vtreat/blob/master/Examples/fit_transform/fit_transform_api.md`, `mkCrossFrameMExperiment` and `prepare.multinomial_plan` for details.

Note: there currently is no `designTreatmentsM`, so `MultinomialOutcomeTreatment$fit()` is implemented in terms of `MultinomialOutcomeTreatment$fit_transform()`

Description

A list of settings and values for `vtreat` multinomial classification fitting. Please see `https://github.com/WinVector/vtreat/blob/master/Examples/fit_transform/fit_transform_api.md`, `mkCrossFrameMExperiment` and `prepare.multinomial_plan` for details.

Usage

`multinomial_parameters(user_params = NULL)`

Arguments

user_params: list of user overrides.

Value

filled out parameter list
Report new/novel appearances of character values.

Description

Report new/novel appearances of character values.

Usage

novel_value_summary(dframe, trackedValues)

Arguments

- **dframe**: Data frame to inspect.
- **trackedValues**: optional named list mapping variables to know values, allows warnings upon novel level appearances (see `track_values`)

Value

frame of novel occurrences

See Also

- `prepare.treatmentplan`
- `track_values`

Examples

```r
set.seed(23525)
zip <- c(NA, paste('z', 1:10, sep = "_"))
N <- 10
d <- data.frame(zip = sample(zip, N, replace=TRUE),
    zip2 = sample(zip, N, replace=TRUE),
    y = runif(N))
dSample <- d[1:5, , drop = FALSE]
trackedValues <- track_values(dSample, c("zip", "zip2"))
novel_value_summary(d, trackedValues)
```
NumericOutcomeTreatment

Stateful object for designing and applying numeric outcome treatments.

Description

Hold settings are results for regression data preparation.

Usage

NumericOutcomeTreatment(
  ..., var_list, outcome_name, cols_to_copy = NULL, params = NULL, imputation_map = NULL
)

Arguments

... not used, force arguments to be specified by name.
var_list Names of columns to treat (effective variables).
outcome_name Name of column holding outcome variable. dframe[[outcome_name]] must be only finite non-missing values.
cols_to_copy list of extra columns to copy.
params parameters list from regression_parameters
imputation_map map from column names to functions of signature f(values: numeric, weights: numeric), simple missing value imputers.

Details

Please see https://github.com/WinVector/vtreat/blob/master/Examples/fit_transform/fit_transform_api.md, mkCrossFrameNExperiment, designTreatmentsN, and prepare.treatmentplan for details.
oneWayHoldout

One way holdout, a splitFunction in the sense of vtreat::buildEvalSets.

Description

Note one way holdout can leak target expected values, so it should not be preferred in nested modeling situations. Also, 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)

patch_columns_into_frame

Patch columns into data.frame.

Description

Add columns from new_frame into old_frame, replacing any columns with matching names in orig_frame with values from new_frame.

Usage

patch_columns_into_frame(orig_frame, new_frame)

Arguments

orig_frame data.frame to patch into.
new_frame data.frame to take replacement columns from.
Examples

```r
orig_frame <- data.frame(x = 1, y = 2)
new_frame <- data.frame(y = 3, z = 4)
patch_columns_into_frame(orig_frame, new_frame)
```

---

### ppCoderC

**Solve a categorical partial pooling problem.**

**Description**


**Usage**

```r
ppCoderC(v, vcol, y, w = NULL)
```

**Arguments**

- `v`: character variable name
- `vcol`: character, independent or input variable
- `y`: logical, dependent or outcome variable to predict
- `w`: row/example weights

**Value**

scored training data column
ppCoderN

Solve a numeric partial pooling problem.

Description


Usage

ppCoderN(v, vcol, y, w = NULL)

Arguments

v character variable name
vcol character, independent or input variable
y numeric, dependent or outcome variable to predict
w row/example weights

Value

scored training data column

prepare

Apply treatments and restrict to useful variables.

Description

Apply treatments and restrict to useful variables.

Usage

prepare(treatmentplan, dframe, ...)

Arguments

treatmentplan Plan built by designTreatmentsC() or designTreatmentsN()
dframe Data frame to be treated
... no additional arguments, declared to forced named binding of later arguments

See Also

prepare.treatmentplan, prepare.simple_plan, prepare.multinomial_plan
prepare.multinomial_plan

Function to apply mkCrossFrameMExperiment treatments.

Description

Please see vignette("MultiClassVtreat",package = "vtreat") https://winvector.github.io/vtreat/articles/MultiClassVtreat.html.

Usage

```r
## S3 method for class 'multinomial_plan'
prepare(
  treatmentplan, 
  dframe,
  ..., 
  pruneSig = NULL, 
  scale = FALSE, 
  doCollar = FALSE, 
  varRestriction = NULL, 
  codeRestriction = NULL, 
  trackedValues = NULL, 
  extracols = NULL, 
  parallelCluster = NULL, 
  use_parallel = TRUE
)
```

Arguments

treatmentplan  multinomial_plan from mkCrossFrameMExperiment.
dframe  new data to process.
...  not used, declared to forced named binding of later arguments
pruneSig  suppress variables with significance above this level
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
codeRestriction  optional list of treated variable codes to restrict to
trackedValues  optional named list mapping variables to know values, allows warnings upon novel level appearances (see track_values)
prepare.simple_plan

extracols extra columns to copy.
parallelCluster (optional) a cluster object created by package parallel or package snow.
use_parallel logical, if TRUE use parallel methods.

Value
prepared data frame.

See Also
mkCrossFrameMExperiment, prepare

prepare.simple_plan Prepare a simple treatment.

Description
Prepare a simple treatment.

Usage
## S3 method for class 'simple_plan'
prepare(treatmentplan, dframe, ...)

Arguments
treatmentplan A simple treatment plan.
dframe data.frame to be treated.
... not used, present for S3 signature consistency.

See Also
design_missingness_treatment, prepare

Examples
d <- wrapr::build_frame(
  "x1", "x2", "x3" |
  1 , 4 , "A" |
  NA , 5 , "B" |
  3 , 6 , NA )

plan <- design_missingness_treatment(d)
prepare(plan, d)
prepare(plan, data.frame(x1=NA, x2=NA, x3="E"))
prepare.treatmentplan Apply treatments and restrict to useful variables.

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. Note: treatmentplan’s are not meant for long-term storage, a warning is issued if the version of vtreat that produced the plan differs from the version running prepare().

Usage

```r
## S3 method for class 'treatmentplan'
prepare(
  treatmentplan,
  dframe,
  ..., 
  pruneSig = NULL,
  scale = FALSE,
  doCollar = FALSE,
  varRestriction = NULL,
  codeRestriction = NULL,
  trackedValues = NULL,
  extracols = NULL,
  parallelCluster = NULL,
  use_parallel = TRUE
)
```

Arguments

treatmentplan Plan built by designTreantmentsC() or designTreatmentsN()
dframe Data frame to be treated
... no additional arguments, declared to forced named binding of later arguments
pruneSig optional if TRUE replace numeric variables with significance above this level
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
prepare.treatmentplan

```r
codeRestriction  
optional list of treated variable codes to restrict to

trackedValues  
optional named list mapping variables to know values, allows warnings upon
novel level appearances (see track_values)

extracols  
extra columns to copy.

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

use_parallel  
logical, if TRUE use parallel methods.

Value

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

See Also

mkCrossFrameCExperiment, mkCrossFrameNExperiment, designTreatmentsC, designTreatmentsN, designTreatmentsZ, prepare

Examples

dTrainN <- data.frame(x = c('a','a','a','b','b','b'),
  z = c(1,2,4,5,6,7),
  y = c(0,0,1,1,0,1))
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 = 0.2)
dTestNTreated <- prepare(treatmentsN, dTestN, pruneSig = 0.2)

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)
dTestCTreated <- prepare(treatmentsC, dTestC)

dTrainZ <- data.frame(x = c('a','a','a','b','b','b'),
  z = c(1,2,3,4,5,6))
dTestZ <- data.frame(x = c('a','b','c',NA),
  z = c(10,20,30,NA))
treatmentsZ <- designTreatmentsZ(dTrainZ, colnames(dTrainZ))
dTrainZTreated <- prepare(treatmentsZ, dTrainZ, codeRestriction = c('lev'))
dTestZTreated <- prepare(treatmentsZ, dTestZ, codeRestriction = c('lev'))
```
**pre_comp_xval**  
*Pre-computed cross-plan (so same split happens each time).*

**Description**  
Pre-computed cross-plan (so same split happens each time).

**Usage**  
pre_comp_xval(nRows, nSplits, splitplan)

**Arguments**
- *nRows*: number of rows to split (integer >1).
- *nSplits*: number of groups to split into (ignored).
- *splitplan*: split plan to actually use

**Value**
- *splitplan*

**Examples**

```r
p1 <- oneWayHoldout(3,NULL,NULL,NULL)
p2 <- pre_comp_xval(3, 3, p1)
p2(3, 3)
```

---

**print.multinomial_plan**  
*Print treatmentplan.*

**Description**  
Print treatmentplan.

**Usage**

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

**Arguments**
- *x*: treatmentplan
- *...*: additional args (to match general signature).
**Description**

Print treatmentplan.

**Usage**

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

**Arguments**

- `x`: treatmentplan
- `...`: additional args (to match general signature).

**Description**

Print treatmentplan.

**Usage**

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

**Arguments**

- `x`: treatmentplan
- `...`: additional args (to match general signature).

**See Also**

`designTreatmentsC`, `designTreatmentsN`, `designTreatmentsZ`, `prepare.treatmentplan`
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, designTreatmentsZ, prepare.treatmentplan

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

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)
regression_parameters

See Also

kWayCrossValidation, kWayStratifiedY, and makekWayCrossValidationGroupedByColumn

Examples

plan <- kWayStratifiedY(3,2,NULL,NULL)
problemAppPlan(3,plan,TRUE)

regression_parameters  vtreat regression parameters.

Description

A list of settings and values for vtreat regression fitting. Please see https://github.com/WinVector/vtreat/blob/master/Examples/fit_transform/fit_transform_api.md, mkCrossFrameCExperiment, designTreatmentsC, and mkCrossFrameNExperiment, designTreatmentsN, prepare.treatmentplan for details.

Usage

regression_parameters(user_params = NULL)

Arguments

user_params  list of user overrides.

Value

filled out parameter list

rquery_prepare  Materialize a treated data frame remotely.

Description

Materialize a treated data frame remotely.
Usage

rquery_prepare(
  db,
  rqplan,
  data_source,
  result_table_name,
  ..., extracols = NULL,
  temporary = FALSE,
  overwrite = TRUE,
  attempt_nan_inf_mapping = FALSE,
  col_sample = NULL,
  return_ops = FALSE
)

materialize_treated(
  db,
  rqplan,
  data_source,
  result_table_name,
  ..., extracols = NULL,
  temporary = FALSE,
  overwrite = TRUE,
  attempt_nan_inf_mapping = FALSE,
  col_sample = NULL,
  return_ops = FALSE
)

Arguments

db a db handle.
	nplean an query plan produced by as_rquery_plan().

data_source relop, data source (usually a relop_table_source).

result_table_name character, table name to land result in

... force later arguments to bind by name.

extracols extra columns to copy.

temporary logical, if TRUE try to make result temporary.

overwrite logical, if TRUE try to overwrite result.

attempt_nan_inf_mapping logical, if TRUE attempt to map NaN and Infinity to NA/NULL (goot on PostgreSQL, not on Spark).

col_sample sample of data to determine column types.

return_ops logical, if TRUE return operator tree instead of materializing.
run_vtreat_tests

Value
description of treated table.

Functions
  • materialize_treated: old name for rquery_prepare function

See Also
  as_rquery_plan, rqdatatable_prepare

run_vtreat_tests  Run vtreat tests.

Description
For all files with names of the form "^test_.+\.R$" in the package directory unit_tests run all func-
tions with names of the form "^test_+\$" as RUnit tests. Attaches RUnit and pkg, requires RUnit. Stops on error.

Usage
run_vtreat_tests(
  ..., verbose = TRUE,
  package_test_dirs = "unit_tests",
  test_dirs = character(0),
  stop_on_issue = TRUE,
  stop_if_no_tests = TRUE,
  require_RUnit_attached = FALSE,
  require_pkg_attached = TRUE,
  rngKind = "Mersenne-Twister",
  rngNormalKind = "Inversion"
)

Arguments
  ... not used, force later arguments to bind by name.
  verbose logical, if TRUE print more.
  package_test_dirs directory names to look for in the installed package.
  test_dirs paths to look for tests in.
  stop_on_issue logical, if TRUE stop after errors or failures.
  stop_if_no_tests logical, if TRUE stop if no tests were found.
require_RUnit_attached
  logical, if TRUE require RUnit be attached before testing.
require_pkg_attached
  logical, if TRUE require pkg be attached before testing.
rngKind
  pseudo-random number generator method name.
rngNormalKind
  pseudo-random normal generator method name.

Value

RUnit test results (invisible).

---

solveIsotone: Solve for best single-direction (non-decreasing or non-increasing) fit.

Description

Return a vector of length y that is a function of x (differs at must where x differs) obeying the either the same order constraints or the opposite order constraints as x. This vector is picked as close to y (by square-distance) as possible.

Usage

solveIsotone(varName, x, y, w = NULL)

Arguments

varName: character, name of variable
x: numeric, factor, or character input (not empty, no NAs).
y: numeric (same length as x no NAs), output to match
w: numeric positive, same length as x (weights, can be NULL)

Details

Please see https://github.com/WinVector/vtreat/blob/master/extras/MonotoneCoder.md.

Value

isotonicly adjusted y (non-decreasing)

Examples

if(requireNamespace("isotone", quietly = TRUE)) {
  solveIsotone('y', 1:3, c(1,2,1))
}
solveNonDecreasing

Solve for best non-decreasing fit using isotone regression (from the "isotone" package http://CRAN.R-project.org/package=isotone).

Description

Return a vector of length y that is a function of x (differs at must where x differs) obeying the same order constraints as x. This vector is picked as close to y (by square-distance) as possible.

Usage

solveNonDecreasing(varName, x, y, w = NULL)

Arguments

- varName: character, name of variable
- x: numeric, factor, or character input (not empty, no NAs).
- y: numeric or castable to such (same length as x no NAs), output to match
- w: numeric positive, same length as x (weights, can be NULL)

Details

Please see https://github.com/WinVector/vtreat/blob/master/extras/MonotoneCoder.md.

Value

isotonically adjusted y (non-decreasing)

Examples

```r
if(requireNamespace("isotone", quietly = TRUE)) {
  solveNonDecreasing('v', 1:3, c(1,2,1))
}
```
solveNonIncreasing  Solve for best non-increasing fit.

Description

Return a vector of length y that is a function of x (differs at must where x differs) obeying the opposite order constraints as x. This vector is picked as close to y (by square-distance) as possible.

Usage

solveNonIncreasing(varName, x, y, w = NULL)

Arguments

varName character, name of variable
x numeric, factor, or character input (not empty, no NAs).
y numeric (same length as x no NAs), output to match
w numeric positive, same length as x (weights, can be NULL)

Details

Please see https://github.com/WinVector/vtreat/blob/master/extras/MonotoneCoder.md.

Value

isotonicly adjusted y (non-decreasing)

Examples

if(requireNamespace("isotone", quietly = TRUE)) {
  solveNonIncreasing('v', 1:3, c(2,1))
}

solve_piecewise

Solve as piecewise linear problem, numeric target.

Description

Return a vector of length y that is a piecewise function of x. This vector is picked as close to y (by square-distance) as possible for a set of x-only determined cut-points. Cross-validates for a good number of segments.

Usage

solve_piecewise(varName, x, y, w = NULL)

Arguments

- varName: character, name of variable
- x: numeric input (not empty, no NAs).
- y: numeric or castable to such (same length as x no NAs), output to match
- w: numeric positive, same length as x (weights, can be NULL)

Value

segmented y prediction

solve_piecewisec

Solve as piecewise logit problem, categorical target.

Description

Return a vector of length y that is a piecewise function of x. This vector is picked as close to y (by square-distance) as possible for a set of x-only determined cut-points. Cross-validates for a good number of segments.

Usage

solve_piecewisec(varName, x, y, w = NULL)

Arguments

- varName: character, name of variable
- x: numeric input (not empty, no NAs).
- y: numeric or castable to such (same length as x no NAs), output to match
- w: numeric positive, same length as x (weights, can be NULL)

Value

segmented y prediction
spline_variable

Spline variable numeric target.

Description
Return a spline approximation of data.

Usage
spline_variable(varName, x, y, w = NULL)

Arguments
varName character, name of variable
x numeric input (not empty, no NAs).
y numeric or castable to such (same length as x no NAs), output to match
w numeric positive, same length as x (weights, can be NULL)

Value
spline y prediction

spline_variablec

Spline variable categorical target.

Description
Return a spline approximation of the change in log odds.

Usage
spline_variablec(varName, x, y, w = NULL)

Arguments
varName character, name of variable
x numeric input (not empty, no NAs).
y numeric or castable to such (same length as x no NAs), output to match
w numeric positive, same length as x (weights, can be NULL)

Value
spline y prediction
Build a square windows variable, numeric target.

Description

Build a square moving average window (KNN in 1d). This is a high-frequency feature.

Usage

square_window(varName, x, y, w = NULL)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>varName</td>
<td>character, name of variable</td>
</tr>
<tr>
<td>x</td>
<td>numeric input (not empty, no NAs).</td>
</tr>
<tr>
<td>y</td>
<td>numeric or castable to such (same length as x no NAs), output to match</td>
</tr>
<tr>
<td>w</td>
<td>numeric positive, same length as x (weights, can be NULL) IGNORED</td>
</tr>
</tbody>
</table>

Value

segmented y prediction

Examples

d <- data.frame(x = c(NA, 1:6), y = c(0, 0, 0, 1, 1, 0, 0))
square_window("v", d$x, d$y)

Build a square windows variable, categorical target.

Description

Build a square moving average window (KNN in 1d). This is a high-frequency feature. Approximation of the change in log odds.

Usage

square_windowc(varName, x, y, w = NULL)
Arguments

<table>
<thead>
<tr>
<th>varName</th>
<th>character, name of variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>numeric input (not empty, no NAs).</td>
</tr>
<tr>
<td>y</td>
<td>numeric or castable to such (same length as x no NAs), output to match</td>
</tr>
<tr>
<td>w</td>
<td>numeric positive, same length as x (weights, can be NULL) IGNORED</td>
</tr>
</tbody>
</table>

Value

segmented y prediction

Examples

```r
d <- data.frame(x = c(NA, 1:6), y = c(0, 0, 0, 1, 1, 0, 0))
square_window("v", d$x, d$y)
```

track_values | Track unique character values for variables.

Description

Builds lists of observed unique character values of varlist variables from the data frame.

Usage

```r
track_values(dframe, varlist)
```

Arguments

<table>
<thead>
<tr>
<th>dframe</th>
<th>Data frame to learn treatments from (training data), must have at least 1 row.</th>
</tr>
</thead>
<tbody>
<tr>
<td>varlist</td>
<td>Names of columns to treat (effective variables).</td>
</tr>
</tbody>
</table>

Value

named list of values seen.

See Also

prepare.treatmentplan, novel_value_summary
Examples

```r
set.seed(23525)
zip <- c(NA, paste('z', 1:100, sep = "_"))
N <- 500
d <- data.frame(zip = sample(zip, N, replace=TRUE),
    zip2 = sample(zip, N, replace=TRUE),
    y = runif(N))
dSample <- d[1:300, , drop = FALSE]
tplan <- designTreatmentsN(dSample,
    c("zip", "zip2"), "y",
    verbose = FALSE)
trackedValues <- track_values(dSample, c("zip", "zip2"))
# don't normally want to catch warnings, # doing it here as this is an example # and must not have unhandled warnings.
tryCatch(
    prepare(tplan, d, trackedValues = trackedValues),
    warning = function(w) { cat(paste(w, collapse = "\n")) })
```

UnsupervisedTreatment

Stateful object for designing and applying unsupervised treatments.

Description

Hold settings are results for unsupervised data preparation.

Usage

```r
UnsupervisedTreatment(
    ..., 
    var_list,
    cols_to_copy = NULL,
    params = NULL,
    imputation_map = NULL
)
```

Arguments

... not used, force arguments to be specified by name.
var_list Names of columns to treat (effective variables).
cols_to_copy list of extra columns to copy.
params parameters list from unsupervised_parameters
imputation_map map from column names to functions of signature f(values: numeric, weights: numeric), simple missing value imputers.
Details

Please see https://github.com/WinVector/vtreat/blob/master/Examples/fit_transform/fit_transform_api.md, designTreatmentsZ and prepare.treatmentplan for details.

Note: for UnsupervisedTreatment fit_transform(d) is implemented as fit(d)$transform(d).

unsupervised_parameters

vtreat unsupervised parameters.

Description

A list of settings and values for vtreat unsupervised fitting. Please see https://github.com/WinVector/vtreat/blob/master/Examples/fit_transform/fit_transform_api.md, designTreatmentsZ, and prepare.treatmentplan for details.

Usage

unsupervised_parameters(user_params = NULL)

Arguments

user_params list of user overrides.

Value

filled out parameter list

value_variables_C

Value variables for prediction a categorical outcome.

Description

Value variables for prediction a categorical outcome.

Usage

value_variables_C(dframe, varlist, outcomename, outcometarget, ..., weights = c(), minFraction = 0.02, smFactor = 0, ...
value_variables_C

rareCount = 0,
rareSig = 1,
collarProb = 0,
scale = FALSE,
doCollar = FALSE,
splitFunction = NULL,
ncross = 3,
forceSplit = FALSE,
catScaling = TRUE,
verbose = FALSE,
parallelCluster = NULL,
use_parallel = TRUE,
customCoders = list(c.PiecewiseV.num = vtreat::solve_piecewisec, n.PiecewiseV.num = vtreat::solve_piecewise, c.knearest.num = vtreat::square_windowc, n.knearest.num = vtreat::square_window),
codeRestriction = c("PiecewiseV", "knearest", "clean", "isBAD", "catB", "catP"),
missingness_imputation = NULL,
imputation_map = 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 dframe[[outcomename]]==outcometarget at least twice and dframe[[outcomename]]!=outcometarget 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.treatmentplan.
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 collarProb during treatment design.
splitFunction (optional) see vtreat::buildEvalSets.
value_variables_N

ncross optional scalar>=2 number of cross-validation rounds to design.
forceSplit logical, if TRUE force cross-validated significance calculations on all variables.
catScaling optional, if TRUE use glm() linkspace, if FALSE use lm() for scaling.
verbose if TRUE print progress.
parallelCluster (optional) a cluster object created by package parallel or package snow.
use_parallel logical, if TRUE use parallel methods.
customCoders additional coders to use for variable importance estimate.
codeRestriction codes to restrict to for variable importance estimate.
missingness_imputation function of signature f(values: numeric, weights: numeric), simple missing value imputer.
imputation_map map from column names to functions of signature f(values: numeric, weights: numeric), simple missing value imputers.

Value
table of variable valuations

value_variables_N Value variables for prediction a numeric outcome.

Description
Value variables for prediction a numeric outcome.

Usage
value_variables_N(
dframe,
varlist,
outcomename,
..., 
weights = c(),
minFraction = 0.02,
smFactor = 0,
rareCount = 0,
rareSig = 1,
collarProb = 0,
scale = FALSE,
doCollar = FALSE,
splitFunction = NULL,
ncross = 3,
forceSplit = FALSE,
verbose = FALSE,
parallelCluster = NULL,
use_parallel = TRUE,
customCoders = list(c.PiecewiseV.num = vtreat::solve_piecewisec,
                    n.PiecewiseV.num = vtreat::solve_piecewise,
c.knearest.num = vtreat::square_windowc,
                  n.knearest.num = vtreat::square_window),
codeRestriction = c("PiecewiseV", "knearest", "clean", "isBAD", "catB", "catP"),
missingness_imputation = NULL,
imputation_map = 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).
outcome Name of column holding outcome variable. dframe[[outcome]] must be
        only finite non-missing values and there must be a cut such that dframe[[outcome]]
        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.treatmentplan.
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 collarProb during treatment design.
splitFunction (optional) see vtreat::buildEvalSets.
ncross optional scalar>2 number of cross-validation rounds to design.
forceSplit logical, if TRUE force cross-validated significance calculations on all variables.
verbose if TRUE print progress.
parallelCluster (optional) a cluster object created by package parallel or package snow.
use_parallel logical, if TRUE use parallel methods.
customCoders additional coders to use for variable importance estimate.
codeRestriction codes to restrict to for variable importance estimate.
missingness_imputation

function of signature $f$(values: numeric, weights: numeric), simple missing value imputer.

imputation_map

map from column names to functions of signature $f$(values: numeric, weights: numeric), simple missing value imputers.

Value

table of variable valuations

---

variable_values

*Return variable evaluations.*

---

Description

Return variable evaluations.

Usage

variable_values(sf)

Arguments

sf

scoreFrame from vtreat treatments

Value

per-original variable evaluations

---

vnames

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

---

Description

New treated variable names from a treatmentplan$ treatment item.

Usage

vnames(x)

Arguments

x

vtreatment item

See Also

designTreatmentsC designTreatmentsN designTreatmentsZ
Original variable name from a treatmentplan$ treatment item.

Description
Original variable name from a treatmentplan治疗方法 item.

Usage
vorig(x)

Arguments
x vtreatment item.

See Also
designTreatmentsC designTreatmentsN designTreatmentsZ

vtreat: A Statistically Sound 'data.frame' Processor/Conditioner

Description
A 'data.frame' processor/conditioner that prepares real-world data for predictive modeling in a statistically sound manner. 'vtreat' 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, and new categorical levels (levels seen during application, but not during training). 'vtreat::prepare' should be used as you would use 'model.matrix'.

Details
For more information:

- vignette('vtreat',package='vtreat')
- vignette(package='vtreat')
- Website: https://github.com/WinVector/vtreat
Index

as_rquery_plan, 3, 43
BinomialOutcomeTreatment, 4
buildEvalSets, 5
center_scale, 7
classification_parameters, 8
design_missingness_treatment, 15, 35
designTreatmentsC, 5, 8, 9, 13, 15, 20, 22, 27, 37, 39–41, 56, 57
designTreatmentsN, 11, 11, 15, 22, 25, 27, 30, 37, 39–41, 56, 57
designTreatmentsZ, 11, 13, 13, 37, 39, 40, 52, 56, 57
format.vtreatment, 16
getSplitPlanAppLabels, 17
kWayCrossValidation, 6, 17, 17, 41
kWayStratifiedY, 6, 17, 18, 41
kWayStratifiedYReplace, 19
makekWayCrossValidationGroupedByColumn, 6, 17, 20, 41
materialize_treated (rquery_prepare), 41
mkCrossFrameCExperiment, 5, 8, 9, 11, 20, 37, 41
mkCrossFrameMExperiment, 23, 28, 35
mkCrossFrameNExperiment, 11, 13, 25, 30, 37, 41
multinomial_parameters, 28
MultinomialOutcomeTreatment, 27
novel_value_summary, 29, 50
NumericOutcomeTreatment, 30
oneWayHoldout, 31
patch_columns_into_frame, 31
ppCoderC, 32
ppCoderN, 33
pre_comp_xval, 38
prepare, 33, 35, 37
prepare.multinomial_plan, 24, 25, 28, 33, 34
prepare.simple_plan, 16, 33, 35
prepare.treatmentplan, 5, 8, 10–15, 22, 26, 27, 29, 30, 33, 36, 39–41, 50, 52, 53, 55
print.multinomial_plan, 38
print.simple_plan, 39
print.treatmentplan, 39
print.vtreatment, 40
problemAppPlan, 40
regression_parameters, 41
rqdatatable_prepare, 43
rquery_prepare, 3, 41
run_vtreat_tests, 43
solve_piecewise, 47
solve_piecewisec, 47
solveIsotone, 44
solveNonDecreasing, 45
solveNonIncreasing, 46
spline_variable, 48
spline_variablec, 48
square_window, 49
square_windowc, 49
track_values, 29, 34, 37, 50
unsupervised_parameters, 52
UnsupervisedTreatment, 51
value_variables_C, 52
value_variables_N, 54
variable_values, 56
vnames, 56
vorig, 57
INDEX

vtreat, 57