Package 'mltools'

October 13, 2022

Type Package

Title Machine Learning Tools

Version 0.3.5

Author Ben Gorman

Maintainer Ben Gorman <bgorman@GormAnalysis.com>

Description A collection of machine learning helper functions, particularly assisting in the Exploratory Data Analysis phase. Makes heavy use of the 'data.table' package for optimal speed and memory efficiency. Highlights include a versatile bin_data() function, sparsify() for converting a data.table to sparse matrix format with one-hot encoding, fast evaluation metrics, and empirical_cdf() for calculating empirical Multivariate Cumulative Distribution Functions.

License MIT + file LICENSE

URL https://github.com/ben519/mltools

BugReports https://github.com/ben519/mltools/issues

LazyData TRUE

RoxygenNote 6.0.1

Imports data.table(>= 1.9.7), Matrix, methods, stats

Suggests testthat

NeedsCompilation no

Repository CRAN

Date/Publication 2018-05-12 03:12:45 UTC

R topics documented:

`alien.test` *Alien test dataset*

Description

A dataset describing features of living beings

Usage

```
alien.test
```

Format

A data.table with 8 rows and 5 variables:

SkinColor Skin color of the individual

IQScore IQ score of the individual

Cat1 Categorical descriptor

Cat2 Categorical descriptor

Cat3 Categorical descriptor

Details

library(data.table)

alien.test <- data.table::data.table(SkinColor=c("white", "green", "brown", "white", "red"), IQScore=c(79, 100, 125, 90, 115), Cat1=c("type4", "type4", "type3", "type1", "type1"), Cat2=c("type5", "type5", "type9", "type8", "type2"), Cat3=c("type2", "type2", "type7", "type4", "type4"))

save(alien.test, file="data/alien_test.rda")

`alien.train` *Alien training dataset*

Description

A dataset describing features of living beings and whether or not they are an alien

Usage

```
alien.train
```

Format

A data.table with 8 rows and 6 variables:

SkinColor Skin color of the individual

IQScore IQ score of the individual

Cat1 Categorical descriptor

Cat2 Categorical descriptor

Cat3 Categorical descriptor

IsAlien Is this being an alien?

Details

library(data.table)

alien.train <- data.table::data.table(SkinColor=c("green", "white", "brown", "white", "blue", "white", "green", "white"), IQScore=c(300, 95, 105, 250, 115, 85, 130, 115), Cat1=c("type1", "type1", "type2", "type4", "type2", "type4", "type1", "type1"), Cat2=c("type1", "type2", "type6", "type5", "type7", "type5", "type2", "type1"), Cat3=c("type4", "type4", "type11", "type2", "type11", "type2", "type4", "type4"), IsAlien=c(TRUE, FALSE, FALSE, TRUE, TRUE, FALSE, TRUE, FALSE))

save(alien.train, file="data/alien_train.rda")

auc_roc *Area Under the ROC Curve*

Description

Calculates Area Under the ROC Curve

Usage

```
auc_roc(preds, actuals, returnDT = FALSE)
```

Arguments

preds	A vector of prediction values
actuals	A vector of actuals values (numeric or ordered factor)
returnDT	If TRUE, a data.table of (FalsePositiveRate, TruePositiveRate) pairs is returned, otherwise AUC ROC score is returned

Details

If `returnDT=FALSE`, returns Area Under the ROC Curve.If `returnDT=TRUE`, returns a data.table object with False Positive Rate and True Positive Rate for plotting the ROC curve.

References

https://en.wikipedia.org/wiki/Receiver_operating_characteristic#Area_under_the_curve

Examples

```
library(data.table)
preds <- c(.1, .3, .3, .9)
actuals <- c(0, 0, 1, 1)
auc_roc(preds, actuals)
auc_roc(preds, actuals, returnDT=TRUE)
```

bin_data *Map a vector of numeric values into bins*

Description

Takes a vector of values and bin parameters and maps each value to an ordered factor whose levels are a set of bins like [0,1), [1,2), [2,3).

Values may be provided as a vector or via a pair of parameters - a data.table object and the name of the column to bin.

Usage

```
bin_data(x = NULL, binCol = NULL, bins = 10, binType = "explicit",
  boundaryType = "lcro]", returnDT = FALSE)
```

Arguments

x	A vector of values or a data.table object
binCol	A column of dt specifying the values to bin
bins	• **integer** specifying the number of bins to generate • **numeric vector** specifying sequential bin boundaries {(x0, x1), (x1, x2), ..., (xn-1, xn)} • **2-column data.frame/data.table** each row defines a bin
binType	• **"explicit"** interpret bins as they are given • **"quantile"** interpret bins as quantiles (empty quantile bins will be discarded)
boundaryType	• **"lcro]"** bins are [left-closed, right-open) except for last bin which is [left-closed, right-closed] • **"lcro)"** bins are [left-closed, right-open) • **"[lorc"** bins are (left-open, right-closed] except for first bin which is [left-closed, right-closed] • **"(lorc"** bins are (left-open, right-closed]
returnDT	If **FALSE**, return an ordered factor of bins corresponding to the values given, else return a data.table object which includes all bins and values (makes a copy of data.table object if given)

Details

This function can return two different types of output, depending on whether returnDT is TRUE or FALSE.

If returnDT=FALSE, returns an ordered factor vector of bins like [1, 2), [-3,-2), ... corresponding to the values which were binned and whose levels correspond to all the generated bins. (Note that empty bins may be present as unused factor levels).

If returnDT=TRUE, returns a data.table object with all values and all bins (including empty bins). If dt is provided instead of vals, a full copy of dt is created and merged with the set of generated bins.

Examples

```
library(data.table)
iris.dt <- data.table(iris)

# custom bins
bin_data(iris.dt, binCol="Sepal.Length", bins=c(4, 5, 6, 7, 8))

# 10 equally spaced bins
bin_data(iris$Petal.Length, bins=10, returnDT=TRUE)
```

```
# make the last bin [left-closed, right-open)
bin_data(c(0,0,1,2), bins=2, boundaryType="lcro)", returnDT=TRUE)

# bin values by quantile
bin_data(c(0,0,0,0,1,2,3,4), bins=4, binType="quantile", returnDT=TRUE)
```

`date_factor` *Date Factor*

Description

Map a vector of dates to a factor at one of these levels "yearmonth", "yearquarter", "quarter", "month"

Usage

```
date_factor(dateVec, type = "yearmonth", minDate = min(dateVec, na.rm =
  TRUE), maxDate = max(dateVec, na.rm = TRUE))
```

Arguments

`dateVec`	A vector of date values
`type`	One of "year", "yearquarter", "yearmonth", "quarter", "month"
`minDate`	(Default = min(dateVec)) When determining factor levels, use this date to set the min level, after coercing dates to the specified `type`. For example, if dateVec = (2016-01-15, 2016-02-15), type = "yearmonth", and minDate = 2016-02-01, the result will be (NA, Feb 2016).
`maxDate`	(Default = max(dateVec)) When determining factor levels, use this date to set the max level. (See minDate, above)

Details

The resulting vector is an ordered factor of the specified `type` (e.g. yearmonth)

Examples

```
library(data.table)
dts <- as.Date(c("2014-1-1", "2015-1-15", "2015-6-1"))
date_factor(dts, type = "yearmonth")
date_factor(dts, type = "yearquarter")
date_factor(
  dateVec = dts,
  type = "yearquarter",
  minDate = as.Date("2015-1-1"),
  maxDate = as.Date("2015-12-31")
)
```

```
date_factor(
  dateVec = as.Date(character(0)),
  type = "yearmonth",
  minDate = as.Date("2016-1-1"),
  as.Date("2016-12-31")
)
```

`empirical_cdf` *Empirical Cumulative Distribution Function*

Description

Given a vector x, calculate P(x <= X) for a set of upper bounds X. Can be applied to a data.table object for multivariate use. That is, calculate P(x <= X, y <= Y, z <= Z, ...)

Usage

```
empirical_cdf(x, ubounds)
```

Arguments

x	Numeric vector or a data.table object for multivariate use.
ubounds	A vector of upper bounds on which to evaluate the CDF. For multivariate version, a data.table whose names correspond to columns of x.

Details

Calculate the empirical CDF of a vector, or data.table with multiple columns for multivariate use.

Examples

```
library(data.table)
dt <- data.table(x=c(0.3, 1.3, 1.4, 3.6), y=c(1.2, 1.2, 3.8, 3.9))
empirical_cdf(dt$x, ubounds=1:4)
empirical_cdf(dt, ubounds=CJ(x = 1:4, y = 1:4))
```

explore_dataset *Explore Dataset*

Description

(Experimental) Automated Exploratory Data Analysis

Usage

```
explore_dataset(dt1, dt2 = NULL, targetCol = NULL, verbose = FALSE)
```

Arguments

dt1	dataset to analyze
dt2	(optional) second dataset to analyze, with the same columns as dt1
targetCol	Name of the column you're trying to model/predict
verbose	Should the exploratory process steps be displayed?

Details

Expirimental. Evaluates and summarizes the data in every column of a data.table. Can identify columns with hierarchical structure and columns with perfectly correlated values.

Examples

```
library(data.table)
explore_dataset(alien.train)
```

exponential_weight *Exponential Weight*

Description

Generate exponential weights

Usage

```
exponential_weight(k, base = exp(1), offset = 0, slope = 0.1)
```

Arguments

k	1-base^(offset-slope*k)
base	1-base^(offset-slope*k)
offset	1-base^(offset-slope*k)
slope	1-base^(offset-slope*k)

Details

Returns a weight based on the formula 1-base^(offset-slope*k)

Examples

```
exponential_weight(1:3, slope=.1)
exponential_weight(1:3, slope=1)
exponential_weight(1:3, slope=10)
```

---

folds *Cross Validation Folds*

Description

Map an object x into equal (or nearly equal) size folds. If x is a positive integer, a vector of FoldIDs of length matching x is returned, otherwise If x is a vector, a matching vector of FoldIDs is returned. If x is a data.table, a list of partitions of x is returned.

Usage

```
folds(x, nfolds = 5L, stratified = FALSE, seed = NULL)
```

Arguments

x	A positive integer, a vector of values or a data.table object
nfolds	How many folds?
stratified	If x is a vector then TRUE or FALSE indicating whether x's split the class's of x proportionally. If x is a data.table then `stratified` should be FALSE or the name of a column in x on which to perform stratification. Note that stratification is implemented for categorical, logical, AND numeric x
seed	Random number seed

Details

Convenient method for mapping an object into equal size folds, potentially with stratification

Examples

```
library(data.table)
folds(8, nfolds=2)
folds(alien.train$IsAlien, nfolds=2)
folds(alien.train$IsAlien, nfolds=2, stratified=TRUE, seed=2016)
folds(alien.train$IQScore, nfolds=2, stratified=TRUE, seed=2016)
folds(alien.train, nfolds=2, stratified="IsAlien", seed=2016)
```

geometric_weight *Geometric Weight*

Description

Generate geometric weights

Usage

```
geometric_weight(k, n, r = 1)
```

Arguments

k	r^k/sum(r^(1, 2, ... n))
n	r^k/sum(r^(1, 2, ... n))
r	r^k/sum(r^(1, 2, ... n))

Details

Returns a weight based on the formula r^k/sum(r^seq_len(n)). The sequence of weights for k=1, 2, ..., n sum to 1

Examples

```
geometric_weight(1:3, n=3, r=1)
geometric_weight(1:3, n=3, r=.5)
geometric_weight(1:3, n=3, r=2)
```

gini_impurities *Gini Impurities*

Description

Identify group weighted gini impurities using pairs of columns within a dataset. Can be used to located hierarchical data, or 1-1 correspondences

Usage

```
gini_impurities(dt, wide = FALSE, verbose = FALSE)
```

Arguments

dt	A data.table with at least two columns
wide	Should the results be in wide format?
verbose	Should progress be printed to the screen?

Details

For pairs of columns (Var1, Var2) in a dataset, calculates the weighted gini impurity of Var2 relative to the groups determined by Var1

Examples

```
library(data.table)
gini_impurities(alien.train)
gini_impurities(alien.train, wide=TRUE)
```

---

gini_impurity *Gini Impurity*

Description

Calculates the Gini Impurity of a set

Usage

```
gini_impurity(vals)
```

Arguments

vals	A vector of values. Values can be given as raw instances like c("red", "red", "blue", "green") or as a named vector of class frequencies like c(red=2, blue=1, green=1)

Details

Gini Impurity is a measure of how often a randomly chosen element from a set would be incorrectly labeled if it was randomly labeled according to the distribution of labels in the set.

Examples

```
gini_impurity(c("red", "red", "blue", "green"))
gini_impurity(c(red=2, blue=1, green=1))
```

mcc *Matthews correlation coefficient*

Description

Calculate Matthews correlation coefficient

Usage

```
mcc(preds = NULL, actuals = NULL, TP = NULL, FP = NULL, TN = NULL,
  FN = NULL, confusionM = NULL)
```

Arguments

preds	A vector of prediction values, or a data.frame or matrix of TRUE/FALSE or 1/0 whose columns correspond to the possible classes
actuals	A vector of actuals values, or a data.frame or matrix of TRUE/FALSE or 1/0 whose columns correspond to the possible classes
TP	Count of true positives (correctly predicted 1/TRUE)
FP	Count of false positives (predicted 1/TRUE, but actually 0/FALSE)
TN	Count of true negatives (correctly predicted 0/FALSE)
FN	Count of false negatives (predicted 0/FALSE, but actually 1/TRUE)
confusionM	Confusion matrix whose (i,j) element represents the number of samples with predicted class i and true class j

Details

Calculate Matthews correlation coefficient. Provide either

- preds and actuals or
- TP, FP, TN, and FN
- confusionM

References

https://en.wikipedia.org/wiki/Matthews_correlation_coefficient

Examples

```
preds <- c(1,1,1,0,1,1,0,0)
actuals <- c(1,1,1,1,0,0,0,0)
mcc(preds, actuals)
mcc(actuals, actuals)
mcc(TP=3, FP=2, TN=2, FN=1)

# Multiclass
```

```
preds <- data.frame(
  setosa = rnorm(n = 150),
  versicolor = rnorm(n = 150),
  virginica = rnorm(n = 150)
)
preds <- preds == apply(preds, 1, max)
actuals <- data.frame(
  setosa = rnorm(n = 150),
  versicolor = rnorm(n = 150),
  virginica = rnorm(n = 150)
)
actuals <- actuals == apply(actuals, 1, max)
mcc(preds = preds, actuals = actuals)

# Confusion matrix
mcc(confusionM = matrix(c(0,3,3,3,0,3,3,3,0), nrow = 3))
mcc(confusionM = matrix(c(1,0,0,0,1,0,0,0,1), nrow = 3))
```

`mse` *Mean Square Error*

Description

Calculate Mean-Square Error (Deviation)

For the ith sample, Squared Error is calculated as SE = (prediction - actual)^2. MSE is then mean(squared errors).

Usage

```
mse(preds = NULL, actuals = NULL, weights = 1, na.rm = FALSE)
```

Arguments

`preds`	A vector of prediction values in [0, 1]
`actuals`	A vector of actuals values in 0, 1, or FALSE, TRUE
`weights`	Optional vectors of weights
`na.rm`	Should (prediction, actual) pairs with at least one NA value be ignored?

Details

Calculate Mean-Square Error (Deviation)

References

https://en.wikipedia.org/wiki/Mean_squared_error

Examples

```
preds <- c(1.0, 2.0, 9.5)
actuals <- c(0.9, 2.1, 10.0)
mse(preds, actuals)
```

`msle` *Mean Square Logarithmic Error*

Description

Calculate Mean-Square-Logarithmic Error (Deviation)

For the ith sample, Squared Logarithmic Error is calculated as SLE = (log(prediction + 1) - log(actual + 1))^2. MSE is then mean(squared logarithmic errors). Note the '+1' in the calculation of SLE which avoids taking the logarithm of 0 for data which may include 0s.

Usage

```
msle(preds = NULL, actuals = NULL, weights = 1, na.rm = FALSE)
```

Arguments

`preds`	A vector of prediction values in [0, 1]
`actuals`	A vector of actuals values in 0, 1, or FALSE, TRUE
`weights`	Optional vectors of weights
`na.rm`	Should (prediction, actual) pairs with at least one NA value be ignored?

Details

Calculate Mean-Square-Logarithmic Error (Deviation)

Examples

```
preds <- c(1.0, 2.0, 9.5)
actuals <- c(0.9, 2.1, 10.0)
msle(preds, actuals)
```

one_hot *One Hot Encode*

Description

One-Hot-Encode unordered factor columns of a data.table

Usage

```
one_hot(dt, cols = "auto", sparsifyNAs = FALSE, naCols = FALSE,
  dropCols = TRUE, dropUnusedLevels = FALSE)
```

Arguments

dt	A data.table
cols	Which column(s) should be one-hot-encoded? DEFAULT = "auto" encodes all unordered factor columns
sparsifyNAs	Should NAs be converted to 0s?
naCols	Should columns be generated to indicate the present of NAs? Will only apply to factor columns with at least one NA
dropCols	Should the resulting data.table exclude the original columns which are one-hot-encoded?
dropUnusedLevels	Should columns of all 0s be generated for unused factor levels?

Details

One-hot-encoding converts an unordered categorical vector (i.e. a factor) to multiple binarized vectors where each binary vector of 1s and 0s indicates the presence of a class (i.e. level) of the of the original vector.

Examples

```
library(data.table)

dt <- data.table(
  ID = 1:4,
  color = factor(c("red", NA, "blue", "blue"), levels=c("blue", "green", "red"))
)

one_hot(dt)
one_hot(dt, sparsifyNAs=TRUE)
one_hot(dt, naCols=TRUE)
one_hot(dt, dropCols=FALSE)
one_hot(dt, dropUnusedLevels=TRUE)
```

relative_position *Relative Position*

Description

Scale a vector of values to the range [0, 1] based on rank/position

Usage

```
relative_position(vals)
```

Arguments

vals	vector of values

Details

Values are ranked and then scaled to the range [0, 1]. Ties result in the same relative position (e.g. `relative_position(c(1, 2, 2, 3))` returns the vector `c(0.0 0.5 0.5 1.0)`). NAs remain as NAs.

Examples

```
relative_position(1:10)
relative_position(c(1, 2, 2, 3))
relative_position(c(1, NA, 3, 4))
```

replace_na *Replace NA Values*

Description

Convenience method for returning a copy of a vector such that NA values are substituted with a replacement value

Usage

```
replace_na(x, repl = "auto")
```

Arguments

x	vector of values
repl	what to substitute in place of NAs

Details

Returns a copy of x such that NAs get replaced with a replacement value. Default replacement value is 0.

Examples

```
replace_na(c(1, NA, 1, 0))
```

---

rmse *Root Mean Square Error*

Description

Calculate Root-Mean-Square Error (Deviation)

For the ith sample, Squared Error is calculated as SE = (prediction - actual)^2. RMSE is then sqrt(mean(squared errors)).

Usage

```
rmse(preds = NULL, actuals = NULL, weights = 1, na.rm = FALSE)
```

Arguments

preds	A vector of prediction values in [0, 1]
actuals	A vector of actuals values in 0, 1, or FALSE, TRUE
weights	Optional vectors of weights
na.rm	Should (prediction, actual) pairs with at least one NA value be ignored?

Details

Calculate Root-Mean-Square Error (Deviation)

References

https://en.wikipedia.org/wiki/Root-mean-square_deviation

Examples

```
preds <- c(1.0, 2.0, 9.5)
actuals <- c(0.9, 2.1, 10.0)
rmse(preds, actuals)
```

rmsle *Root Mean Square Logarithmic Error*

Description

Calculate Root-Mean-Square-Logarithmic Error (Deviation)

For the ith sample, Squared Logarithmic Error is calculated as SLE = (log(prediction + 1) - log(actual + 1))^2. RMSLE is then sqrt(mean(squared logarithmic errors)). Note the '+1' in the calculation of SLE which avoids taking the logarithm of 0 for data which may include 0s.

Usage

```
rmsle(preds = NULL, actuals = NULL, weights = 1, na.rm = FALSE)
```

Arguments

preds	A vector of prediction values in [0, 1]
actuals	A vector of actuals values in 0, 1, or FALSE, TRUE
weights	Optional vectors of weights
na.rm	Should (prediction, actual) pairs with at least one NA value be ignored?

Details

Calculate Root-Mean-Square-Logarithmic Error (Deviation)

Examples

```
preds <- c(1.0, 2.0, 9.5)
actuals <- c(0.9, 2.1, 10.0)
rmsle(preds, actuals)
```

roc_scores *ROC scores*

Description

This function provides a way to identify the worst predictions when measuring Area Under the ROC curve. Simply put, the worst predictions are the ones with very low or high relative prediction scores (usually probabilities) which relate to the positive and negative samples respectively.

Usage

```
roc_scores(preds, actuals)
```

Arguments

preds	vector of predictions (need not be in range [0-1] - only order matters)
actuals	vector of actuals - either logical or vector of 1s and 0s

Details

How it works

- First the relative position (between 0 and 1) of each prediction is determined
- Next the mean of actuals is determined
- For samples whose position is on the correct side of the overall mean, 0 is given
- For samples whose position is on the wrong side of the overall mean, its distance from the mean is given

Examples

```
roc_scores(c(1,2,3,4), actuals=c(1,1,0,0))
roc_scores(c(0.1, 0.2, 0.3, 0.4), actuals=c(TRUE, FALSE, TRUE, FALSE))
```

---

set_factor *Set Factor*

Description

Convience method for dealing with factors. Map a list of vectors to a list of factor vectors (1-1 mapping) such that the factor vectors all have the same levels - the unique values of the union of all the vectors in the list. Optionally group all low frequency values into a "_other_" level.

Usage

```
set_factor(vectorList, aggregationThreshold = 0)
```

Arguments

vectorList	A list of values to convert to factors
aggregationThreshold	Values which appear this many times or less will be grouped into the level "_other_"

Examples

```
x <- c("a", "b", "c", "c")
y <- c("a", "d", "d")
set_factor(list(x, y))
set_factor(list(x, y), aggregationThreshold=1)
```

skewness *Skewness*

Description

Calculates the skewness of each field in a data.table

Usage

```
skewness(dt)
```

Arguments

dt	A data.table

Details

Counts the frequency of each value in each column, then displays the results in descending order

Examples

```
library(data.table)
skewness(alien.train)
```

sparsify *Sparsify*

Description

Convert a data.table object into a sparse matrix (with the same number of rows).

Usage

```
sparsify(dt, sparsifyNAs = FALSE, naCols = "none")
```

Arguments

dt	A data.table object
sparsifyNAs	Should NAs be converted to 0s and sparsified?
naCols	• **"none"** Don't generate columns to identify NA values • **"identify"** For each column of dt with an NA value, generate a column in the sparse matrix with 1s indicating NAs. Columns will be named like "color_NA" • **"efficient"** For each column of dt with an NA value, generate a column in the sparse matrix with 1s indicating either NAs or Non NAs - whichever is more memory efficient. Columns will be named like "color_NA" or "color_NotNA"

Details

Converts a data.table object to a sparse matrix (class "dgCMatrix"). Requires the **Matrix** package. All sparsified data is assumed to take on the value 0/FALSE

Data Type | Description & NA handling

numeric | If `sparsifyNAs` = FALSE, only 0s will be sparsified If `sparsifyNAs` = TRUE, 0s and NAs will be sparsified

factor (unordered) | Each level will generate a sparsified binary column Column names are feature_level, e.g. "color_red", "color_blue"

factor (ordered) | Levels are converted to numeric, 1 - NLevels If `sparsifyNAs` = FALSE, NAs will remain as NAs If `sparsifyNAs` = TRUE, NAs will be sparsified

logical | TRUE and FALSE values will be converted to 1s and 0s If `sparsifyNAs` = FALSE, only FALSEs will be sparsified If `sparsifyNAs` = TRUE, FALSEs and NAs will be sparsified

Examples

```
library(data.table)
library(Matrix)

dt <- data.table(
  intCol=c(1L, NA_integer_, 3L, 0L),
  realCol=c(NA, 2, NA, NA),
  logCol=c(TRUE, FALSE, TRUE, FALSE),
  ofCol=factor(c("a", "b", NA, "b"), levels=c("a", "b", "c"), ordered=TRUE),
  ufCol=factor(c("a", NA, "c", "b"), ordered=FALSE)
)

sparsify(dt)
sparsify(dt, sparsifyNAs=TRUE)
sparsify(dt[, list(realCol)], naCols="identify")
sparsify(dt[, list(realCol)], naCols="efficient")
```

Index