Package 'DMwR2'

October 12, 2022

Type Package

Title Functions and Data for the Second Edition of ``Data Mining with R''

Description Functions and data accompanying the second edition of the book ``Data Mining with R, learning with case studies'' by Luis Torgo, published by CRC Press.

Version 0.0.2

Depends R(>= 3.0), methods

Imports xts (>= 0.9-7), zoo (>= 1.7-10), class (>= 7.3-14), rpart (>= 4.1-10), quantmod (>= 0.4-5), dplyr (>= 0.4.3), readr (>= 1.0.0), DBI (>= 0.5)

Date 2016-10-12

URL https://github.com/ltorgo/DMwR2

BugReports https://github.com/ltorgo/DMwR2/issues

License GPL (>= 2)

LazyLoad yes

LazyData yes

NeedsCompilation no

Author Luis Torgo [aut, cre]

Maintainer Luis Torgo <ltorgo@dcc.fc.up.pt>

Repository CRAN

Date/Publication 2016-10-13 00:23:37

R topics documented:

`DMwR2-package` *Functions and data for the second edition of the book "Data Mining with R"*

Description

This package includes functions and data accompanying the book "Data Mining with R, learning with case studies - 2nd Edition" by Luis Torgo, published by CRC Press

Author(s)

Luis Torgo

Maintainer: Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Torgo, L. (2016) *Data Mining using R: learning with case studies, second edition*, Chapman & Hall/CRC (ISBN-13: 978-1482234893).

http://ltorgo.github.io/DMwR2

algae *Training data for predicting algae blooms*

Description

This data set contains observations on 11 variables as well as the concentration levels of 7 harmful algae. Values were measured in several European rivers. The 11 predictor variables include 3 contextual variables (season, size and speed) describing the water sample, plus 8 chemical concentration measurements.

Usage

```
algae
```

Format

A data frame with 200 observations and 18 columns.

Source

ERUDIT http://www.erudit.de/ - European Network for Fuzzy Logic and Uncertainty Modelling in Information Technology.

algae.sols *The solutions for the test data set for predicting algae blooms*

Description

This data set contains the values of the 7 harmful algae for the 140 test observations in the test set `test.algae`.

Usage

```
algae.sols
```

Format

A data frame with 140 observations and 7 columns.

Source

ERUDIT http://www.erudit.de/ - European Network for Fuzzy Logic and Uncertainty Modelling in Information Technology.

`centralImputation` *Fill in NA values with central statistics*

Description

This function fills in any NA value in all columns of a data frame with the statistic of centrality (given by the function `centralvalue()`) of the respective column.

Usage

```
centralImputation(data)
```

Arguments

`data`	The data frame

Value

A new data frame with no NA values

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Torgo, L. (2016) *Data Mining using R: learning with case studies, second edition*, Chapman & Hall/CRC (ISBN-13: 978-1482234893).

http://ltorgo.github.io/DMwR2

See Also

`knnImputation`, `centralValue`, `complete.cases`, `na.omit`

Examples

```
data(algae,package="DMwR2")
cleanAlgae <- centralImputation(algae)
summary(cleanAlgae)
```

centralValue *Obtain statistic of centrality*

Description

This function obtains a statistic of centrality of a variable given a sample of its values.

Usage

```
centralValue(x, ws = NULL)
```

Arguments

x	A vector of values (the sample).
ws	A vector of case weights (defaulting to NULL, i.e. no case weights).

Details

If the variable is numeric it returns de median of the given sample, if it is a factor it returns the mode. In other cases it tries to convert to a factor and then returns the mode.

Value

A number if the variable is numeric. A string with the name of the most frequent nominal value, otherwise.

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Torgo, L. (2016) *Data Mining using R: learning with case studies, second edition*, Chapman & Hall/CRC (ISBN-13: 978-1482234893).

http://ltorgo.github.io/DMwR2

See Also

mean, median

Examples

```
# An example with numerical data
x <- rnorm(100)
centralValue(x)
# An example with nominal data
y <-
factor(sample(1:10,200,replace=TRUE),levels=1:10,labels=paste('v',1:10,sep=''))
centralValue(y)
```

`createEmbedDS` *Creates an embeded data set from an univariate time series*

Description

Function for creating and embeded data set from a univariate time series given an embed size

Usage

```
createEmbedDS(s, emb=4)
```

Arguments

`s`	A univariate time series (can be a numeric vector or a xts object)
`emb`	The size of the embed for creating the data set (defaults to 4)

Details

The function creates a data set corresponding to the embed of a certain size of a given univariate time series.

For instance for an embed of size 3 each row of the data set will contain the value of the series at time t, t-1 and t-2.

Value

Either a matrix or a multivariate xts, depending on whether the input series is a numberic vector or a univariate xts, respectively.

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Torgo, L. (2016) *Data Mining using R: learning with case studies, second edition*, Chapman & Hall/CRC (ISBN-13: 978-1482234893).

http://ltorgo.github.io/DMwR2

See Also

`embed`

Examples

```
## A simple example with a random time series
x <- rnorm(100)
head(x)
dataSet <- createEmbedDS(x,emb=5)
head(dataSet)
```

`dist.to.knn` *An auxiliary function of* `lofactor()`

Description

This function returns an object in which columns contain the indices of the first k neighbors followed by the distances to each of these neighbors.

Usage

```
dist.to.knn(dataset, neighbors)
```

Arguments

`dataset`	A data set that will be internally coerced into a matrix.
`neighbors`	The number of neighbours.

Details

This function is strongly based on the code provided by Acuna et. al. (2009) for the previously available `dprep` package.

Value

A matrix

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Acuna, E., and Members of the CASTLE group at UPR-Mayaguez, (2009). *dprep: Data preprocessing and visualization functions for classification*. R package version 2.1.

Torgo, L. (2016) *Data Mining using R: learning with case studies, second edition*, Chapman & Hall/CRC (ISBN-13: 978-1482234893).

http://ltorgo.github.io/DMwR2

See Also

`lofactor`

GSPC *A set of daily quotes for SP500*

Description

This is a `xts` object containing the daily quotes of the SP500 sotck index from 1970-01-02 till 2009-09-15 (10,022 daily sessions). For each day information is given on the Open, High, Low and Close prices, and also for the Volume and Adjusted close price.

Usage

```
GSPC
```

Format

A `xts` object with a data matrix with 10,022 rows and 6 columns.

Source

Yahoo Finance

kNN *k-Nearest Neighbour Classification*

Description

This function provides a formula interface to the existing `knn()` function of package `class`. On top of this type of convinient interface, the function also allows standardization of the given data.

Usage

```
kNN(form, train, test, stand = TRUE, stand.stats = NULL, ...)
```

Arguments

`form`	An object of the class `formula` describing the functional form of the classification model.
`train`	The data to be used as training set.
`test`	The data set for which we want to obtain the k-NN classification, i.e. the test set.
`stand`	A boolean indicating whether the training data should be previously normalized before obtaining the k-NN predictions (defaults to TRUE).

stand.stats — This argument allows the user to supply the centrality and spread statistics that will drive the standardization. If not supplied they will default to the statistics used in the function scale(). If supplied they should be a list with two components, each beig a vector with as many positions as there are columns in the data set. The first vector should contain the centrality statistics for each column, while the second vector should contain the spread statistc values.

... — Any other parameters that will be forward to the knn() function of package class.

Details

This function is essentially a convenience function that provides a formula-based interface to the already existing knn() function of package class. On top of this type of interface it also incorporates some facilities in terms of standardization of the data before the k-nearest neighbour classification algorithm is applied. This algorithm is based on the distances between observations, which are known to be very sensitive to different scales of the variables and thus the usefulness of standardization.

Value

The return value is the same as in the knn() function of package class. This is a factor of classifications of the test set cases.

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Torgo, L. (2016) *Data Mining using R: learning with case studies, second edition*, Chapman & Hall/CRC (ISBN-13: 978-1482234893).

http://ltorgo.github.io/DMwR2

See Also

knn, knn1, knn.cv

Examples

```
## A small example with the IRIS data set
data(iris)

## Split in train + test set
idxs <- sample(1:nrow(iris),as.integer(0.7*nrow(iris)))
trainIris <- iris[idxs,]
testIris <- iris[-idxs,]

## A 3-nearest neighbours model with no standardization
nn3 <- kNN(Species ~ .,trainIris,testIris,stand=FALSE,k=3)
```

```
## The resulting confusion matrix
table(testIris[,'Species'],nn3)

## Now a 5-nearest neighbours model with standardization
nn5 <- kNN(Species ~ .,trainIris,testIris,stand=TRUE,k=5)

## The resulting confusion matrix
table(testIris[,'Species'],nn5)
```

---

knneigh.vect *An auxiliary function of* lofactor()

Description

Function that returns the distance from a vector x to its k-nearest-neighbors in the matrix data

Usage

```
knneigh.vect(x, data, k)
```

Arguments

x	An observation.
data	A data set that will be internally coerced into a matrix.
k	The number of neighbours.

Details

This function is strongly based on the code provided by Acuna et. al. (2009) for the previously available dprep package.

Value

A vector.

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Torgo, L. (2016) *Data Mining using R: learning with case studies, second edition*, Chapman & Hall/CRC (ISBN-13: 978-1482234893).

http://ltorgo.github.io/DMwR2

See Also

lofactor

knnImputation *Fill in NA values with the values of the nearest neighbours*

Description

Function that fills in all NA values using the k Nearest Neighbours of each case with NA values. By default it uses the values of the neighbours and obtains an weighted (by the distance to the case) average of their values to fill in the unknows. If meth='median' it uses the median/most frequent value, instead.

Usage

```
knnImputation(data, k = 10, scale = TRUE, meth = "weighAvg",
              distData = NULL)
```

Arguments

data	A data frame with the data set
k	The number of nearest neighbours to use (defaults to 10)
scale	Boolean setting if the data should be scale before finding the nearest neighbours (defaults to T)
meth	String indicating the method used to calculate the value to fill in each NA. Available values are 'median' or 'weighAvg' (the default).
distData	Optionally you may sepecify here a data frame containing the data set that should be used to find the neighbours. This is usefull when filling in NA values on a test set, where you should use only information from the training set. This defaults to NULL, which means that the neighbours will be searched in data

Details

This function uses the k-nearest neighbours to fill in the unknown (NA) values in a data set. For each case with any NA value it will search for its k most similar cases and use the values of these cases to fill in the unknowns.

If meth='median' the function will use either the median (in case of numeric variables) or the most frequent value (in case of factors), of the neighbours to fill in the NAs. If meth='weighAvg' the function will use a weighted average of the values of the neighbours. The weights are given by exp(-dist(k,x) where dist(k,x) is the euclidean distance between the case with NAs (x) and the neighbour k.

Value

A data frame without NA values

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Torgo, L. (2016) *Data Mining using R: learning with case studies, second edition*, Chapman & Hall/CRC (ISBN-13: 978-1482234893).

http://ltorgo.github.io/DMwR2

See Also

centralImputation, centralValue, complete.cases, na.omit

Examples

```
data(algae)
cleanAlgae <- knnImputation(algae)
summary(cleanAlgae)
```

---

lofactor *An implementation of the LOF algorithm*

Description

This function obtain local outlier factors using the LOF algorithm. Namely, given a data set it produces a vector of local outlier factors for each case.

Usage

```
lofactor(data, k)
```

Arguments

data	A data set that will be internally coerced into a matrix.
k	The number of neighbours that will be used in the calculation of the local outlier factors.

Details

This function re-implements the code previously made available in the dprep package (Acuna et. al., 2009) that was removed from CRAN. This code in turn is an implementation of the LOF method by Breunig et. al. (2000). See this reference to understand the full details on how these local outlier factors are calculated for each case in a data set.

Value

The function returns a vector of local outlier factors (numbers). This vector has as many values as there are rows in the original data set.

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Acuna, E., and Members of the CASTLE group at UPR-Mayaguez, (2009). *dprep: Data preprocessing and visualization functions for classification*. R package version 2.1.

Breunig, M., Kriegel, H., Ng, R., and Sander, J. (2000). *LOF: identifying density-based local outliers*. In ACM Int. Conf. on Management of Data, pages 93-104.

Torgo, L. (2016) *Data Mining using R: learning with case studies, second edition*, Chapman & Hall/CRC (ISBN-13: 978-1482234893).

http://ltorgo.github.io/DMwR2

Examples

```
data(iris)
lof.scores <- lofactor(iris[,-5],10)
```

`manyNAs` *Find rows with too many NA values*

Description

Small utility function to obtain the number of the rows in a data frame that have a "large" number of unknown values. "Large" can be defined either as a proportion of the number of columns or as the number in itself.

Usage

```
manyNAs(data, nORp = 0.2)
```

Arguments

`data`	A data frame with the data set.
`nORp`	A number controlling when a row is considered to have too many NA values (defaults to 0.2, i.e. 20% of the columns). If no rows satisfy the constraint indicated by the user, a warning is generated.

Value

A vector with the IDs of the rows with too many NA values. If there are no rows with many NA values and error is generated.

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Torgo, L. (2016) *Data Mining using R: learning with case studies, second edition*, Chapman & Hall/CRC (ISBN-13: 978-1482234893).

http://ltorgo.github.io/DMwR2

See Also

complete.cases, na.omit

Examples

```
data(algae)
manyNAs(algae)
```

---

nrLinesFile *Counts the number of lines of a file*

Description

Function for counting the number of lines of very large text files. Note that it only works on unix-based systems as it uses the wc command line utility

Usage

```
nrLinesFile(f)
```

Arguments

f	A file name (a string)

Details

The function creates a data set corresponding to the embed of a certain size of a given univariate time series.

For instance for an embed of size 3 each row of the data set will contain the value of the series at time t, t-1 and t-2.

Value

An integer

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Torgo, L. (2016) *Data Mining using R: learning with case studies, second edition*, Chapman & Hall/CRC (ISBN-13: 978-1482234893).

http://ltorgo.github.io/DMwR2

See Also

sampleCSV

`outliers.ranking` *Obtain outlier rankings*

Description

This function uses hierarchical clustering to obtain a ranking of outlierness for a set of cases. The ranking is obtained on the basis of the path each case follows within the merging steps of a agglomerative hierarchical clustering method. See the references for further technical details on how these rankings are obtained.

Usage

```
outliers.ranking(data, test.data = NULL, method = "sizeDiff",
                 method.pars = NULL,
                 clus = list(dist = "euclidean",alg = "hclust",
                             meth = "ward.D"),
                 power = 1, verb = F)
```

Arguments

`data`	The data set to be ranked according to outlyingness. This parameter can also be the distance matrix of your additional data set, in case you wish to calculate these distances "outside" of this function.
`test.data`	If a data set is provided in this argument, then the rankings are obtained for these cases and not for the cases provided in the argument `data`. The clustering process driving the obtention of the rankings is carried out on the union of the two sets of data (`data` and `test.data`), but the resulting outlier ranking factors are only for the observations belonging to this set. This parameter defaults to `NULL`.
`method`	The method used to obtain the outlier ranking factors (see the Details section). Defaults to `"sizeDiff"`.
`method.pars`	A list with the parameter values specific to the method selected for obtaining the outlier ranks (see the Details section).

`clus` This is a list that provides several parameters of the clustering process that drives the calculation of the outlier raking factors. If the parameter `data` is not a distance function, then this list should contain a component named `dist` with a value that should be one of the possible values of the parameter `method` the the function `dist()` (see the help of this function for further details). The list should also contain a component named `alg` with the name of the clustering algorithm that should be used. Currently, valid names are either "hclust" (the default) or "diana". Finally, in case the clustering algorithm is "hclust" then the list should also contain a component named `meth` with the name of the agglomerative method to use in the hierarchical clustering algorithm. This should be a valid value of the parameter `method` of the function `hclust()` (check its help page for further details).

`power` Integer value. It allows to raise the distance matrix to some power with the goal of "amplifying" the distance values (defaults to 1).

`verb` Boolean value that determines the level of verbosity of the function (default to FALSE).

Details

This function produces outlier ranking factors for a set of cases. The methodology used for obtaining these factors is described in Section 4.4.1.3 of the book Data Mining with R (Torgo, 2010) and more details can be obtained in Torgo (2007). The methodology is based on the simple idea of using the information provided by an agglomerative hierarchical clustering algorithm to infer the degree of outlyingness of the observations. The basic assumption is that outliers should offer "more resistance" to being clustered, i.e. being merged on large groups of observations.

The function was written to be used with the outcome of the `hclust()` R function that implements several agglomerative clustering methods. Although in theory the methodology could be used with any other agglomerative hierarchical clustering algorithm, the fact is that the code of this implementation strongly depends on the data structures produced by the `hclust()` function. As such if you wish to change the function to be able to use other clustering algorithms you should ensure that the data structures it produces are compatible with the requirements of our function. Specifically, your clustering algorithm should produce a list with a component named `merge` that should be a matrix describing the merging steps of the clustering process (see the help page of the `hclust()` function for a full description of this data structure). This is the only data structure that is required by our function and that is used from the object returned by clustering algorithm. The `diana()` clustering algorithm also produces this type of information and thus can also be used with our function by providing the value "diana" on the component `alg` of the list forming the parameter `clus`.

There are essentially two ways of using this function. The first consists in giving it a data set on the parameter `data` and the function will rank these observations according to their outlyingness. The other consists in specifying two sets of data. One is the set for which you want the outlyingness factors that should be given on the parameter `test.data`. The second set is provided on the `data` parameter and it is used to increase the ammount of data used in the clustering process to improve the statistical reliability of the process.

In the first way of using this function that was described above the user can either supply the data set or the respective distance matrix. If the data set is provided then the user should specify the type of distance metric it should be used to calculate the distances between the observations. This is done by including a distance calculation method in the "dist" component of the list provided in

parameter `clus`. This method should be a valid value of the parameter `method` of the R function `dist()` (see its help for details).

This function currently implements three different methods for obtaing outlier ranking factors from the clustering process. These are: "linear", "sigmoid" and "sizeDiff" (the default). Irrespectively, of this method the outlyingness factor of observation X is obtained by: OF_H(X) = max_i of_i(X), where i represents the different merging steps of the clustering process and it goes from 1 to N-1, where N is the size of the data set to be clustered. The three methods differ in the way they calculate of_i(X) for each merging step. In the "linear" method of_i(X) = i / (N-1) * p(|g|), where g is the group to which X belongs at the merging step i (each merging step involves two groups), and |g| is the size of that group. The function p() is a penalization factor depending on the size of the group. The larger this size the smaller the value of p(), p(s) = I(s < thr) * (1 - (s-1) / (N-2)), where I() is an indicator function and thr is a threshold defined as perc*N. The user should set the value of perc by including a component named "sz.perc" in the list provided in the parameter `method.pars`. In the "sigmoid" method of_i(X) = exp(-2 * (i - (N-1))^2 / (N-1)^2) * p(|g|), where the p() function has the same meaning as in the "linear" method but this time is defined as p(s) = I(s < 2*thr) * (1 - exp(-4 * (s-2*thr)^2 / (2*thr)^2)). Again thr is perc*N and the user must set the value of perc by including a component named "sz.perc" in the list provided in the parameter `method.pars`. Finally, the method "sizeDiff" defines of_i(X) = max (0, (|g_y,i| - |g_x,i|) / (|g_y,i| + |g_x,i|)), where g_y,i and g_x,i are the two groups involved in the merge at step i, and g_x,i is the group which X belongs to. Note that if X belongs to the larger of the two groups this will get X a value of of_i() equals to zero.

Value

The result of this function is a list with four components. Component `rank.outliers` contains a vector with as many positions as there are cases to rank, where position i of the vector contains the rank order of the observation i. Component `prob.outliers` is another vector with the same size this time containing the outlyingness factor (the value of OF_H(X) described in the Details section) of each observation. Component `h` contains the object returned by the clustering process. Finally, component `dist` contains the distance matrix used i nthe clustering process.

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Torgo, L. (2016) *Data Mining using R: learning with case studies, second edition*, Chapman & Hall/CRC (ISBN-13: 978-1482234893).

http://ltorgo.github.io/DMwR2

Torgo, L. (2007) : *Resource-bounded Fraud Detection*, in Progress in Artificial Intelligence, 13th Portuguese Conference on Artificial Intelligence, EPIA 2007, Neves et. al (eds.). LNAI, Springer.

Examples

```
## Some examples with algae frequencies in water samples
data(algae)

## Trying to obtain a reanking of the 200 samples
```

```
o <- outliers.ranking(algae)

## As you may have observed the function complained about some problem
## with the dist() function. The problem is that the algae data frame
## contains columns (the first 3) that are factors and the dist() function
## assumes all numeric data.
## We can solve the problem by calculating the distance matrix "outside"
## using the daisy() function that handles mixed-mode data, as show in
## the code below that requires the R package "cluster" to be available
## dm <- daisy(algae)
## o <- outliers.ranking(dm)

## Now let us check the outlier ranking factors ordered by decreasing
## score of outlyingness
o$prob.outliers[o$rank.outliers]

## Another example with detection of fraudulent transactions
data(sales)

## trying to obtain the outlier ranks for the set of transactions of a
## salesperson regarding one particular product, taking into
## consideration the overall existing transactions of that product
s <- sales[sales$Prod == 'p1',c(1,3:4)]  # transactions of product p1
tr <- na.omit(s[s$ID != 'v431',-1])      # all except salesperson v431
ts <- na.omit(s[s$ID == 'v431',-1])

o <- outliers.ranking(data=tr,test.data=ts,
         clus=list(dist='euclidean',alg='hclust',meth='average'))
# The outlyingness factor of the transactions of this salesperson
o$prob.outliers
```

`reachability` *An auxiliary function of* `lofactor()`

Description

This function computes the reachability measure for each instance of a dataset. This result is used later to compute the Local Outlyingness Factor.

Usage

```
reachability(distdata, k)
```

Arguments

`distdata`	The matrix of distances.
`k`	The number of neighbors.

Details

This function is strongly based on the code provided by Acuna et. al. (2009) for the previously available `dprep` package.

Value

A vector.

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Acuna, E., and Members of the CASTLE group at UPR-Mayaguez, (2009). *dprep: Data preprocessing and visualization functions for classification*. R package version 2.1.

Torgo, L. (2016) *Data Mining using R: learning with case studies, second edition*, Chapman & Hall/CRC (ISBN-13: 978-1482234893).

http://ltorgo.github.io/DMwR2

See Also

`lofactor`

---

rpartXse *Obtain a tree-based model*

Description

This function is based on the tree-based framework provided by the `rpart` package (Therneau et. al. 2010). It basically, integrates the tree growth and tree post-pruning in a single function call. The post-pruning phase is essentially the 1-SE rule described in the CART book (Breiman et. al. 1984).

Usage

```
rpartXse(form, data, se = 1, cp = 0, minsplit = 6, verbose = F, ...)
```

Arguments

`form`	A formula describing the prediction problem
`data`	A data frame containg the training data to be used to obtain the tree-based model
`se`	A value with the number of standard errors to use in the post-pruning of the tree using the SE rule (defaults to 1)
`cp`	A value that controls the stopping criteria used to stop the initial tree growth (defaults to 0)

minsplit	A value that controls the stopping criteria used to stop the initial tree growth (defaults to 6)
verbose	The level of verbosity of the function (defaults to F)
...	Any other arguments that are passed to the rpart() function

Details

The x-SE rule for tree post-pruning is based on the cross-validation estimates of the error of the sub-trees of the initially grown tree, together with the standard errors of these estimates. These values are used to select the final tree model. Namely, the selected tree is the smallest tree with estimated error less than the B+x*SE, where B is the lowest estimate of error and SE is the standard error of this B estimate.

Value

A rpart object

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Therneau, T. M. and Atkinson, B.; port by Brian Ripley. (2010). *rpart: Recursive Partitioning*. R package version 3.1-46.

Breiman, L., Friedman, J., Olshen, R., and Stone, C. (1984). *Classification and regression trees*. Statistics/Probability Series. Wadsworth & Brooks/Cole Advanced Books & Software.

Torgo, L. (2016) *Data Mining using R: learning with case studies, second edition*, Chapman & Hall/CRC (ISBN-13: 978-1482234893).

http://ltorgo.github.io/DMwR2

See Also

rt.prune, rpart, prune.rpart

Examples

```
data(iris)
tree <- rpartXse(Species ~ ., iris)
tree

## A visual representation of the classification tree
## Not run:
prettyTree(tree)

## End(Not run)
```

rt.prune *Prune a tree-based model using the SE rule*

Description

This function implements the SE post pruning rule described in the CART book (Breiman et. al., 1984)

Usage

```
rt.prune(tree, se = 1, verbose = T, ...)
```

Arguments

tree	An rpart object
se	The value of the SE threshold (defaulting to 1)
verbose	The level of verbosity (defaulting to T)
...	Any other arguments passed to the function prune.rpart()

Details

The x-SE rule for tree post-pruning is based on the cross-validation estimates of the error of the sub-trees of the initially grown tree, together with the standard errors of these estimates. These values are used to select the final tree model. Namely, the selected tree is the smallest tree with estimated error less than the B+x*SE, where B is the lowest estimate of error and SE is the standard error of this B estimate.

Value

A rpart object

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Breiman, L., Friedman, J., Olshen, R., and Stone, C. (1984). *Classification and regression trees*. Statistics/Probability Series. Wadsworth & Brooks/Cole Advanced Books & Software.

Torgo, L. (2016) *Data Mining using R: learning with case studies, second edition*, Chapman & Hall/CRC (ISBN-13: 978-1482234893).

http://ltorgo.github.io/DMwR2

See Also

rt.prune, rpart, prune.rpart

Examples

```
data(iris)
tree <- rpartXse(Species ~ ., iris)
tree

## A visual representation of the classification tree
## Not run:
prettyTree(tree)

## End(Not run)
```

`sales` *A data set with sale transaction reports*

Description

This data frame contains 401,146 transaction reports. Each report is made by a salesperson identified by an ID and reports the quantity sold of some product. The data set caontins information on 5 variables: ID (salesperson ID), Prod (product ID), Quant (the sold quantity), Val (the reported value of the transaction) and Insp (a factor containing information on a inspection of the report with possible values 'ok',' fraud' or 'unkn').

Usage

```
sales
```

Format

A data frame with 401,146 rows and 5 columns

Source

Undisclosed

`sampleCSV` *Drawing a random sample of lines from a CSV file*

Description

Function for obtaining a random sample of lines from a very large CSV file, whitout having to load in the full data into memory. Targets situations where the full data does not fit in the computer memory so usage of the standard `sample` function is not possible.

Usage

```
sampleCSV(file, percORn, nrLines, header=TRUE, mxPerc=0.5)
```

Arguments

file	A file name (a string)
percORn	Either the percentage of number of rows of the file or the actual number of rows, the sample should have
nrLines	Optionally you may indicate the number of rows of the file if you know it beforehand, otherwise the function will count them for you
header	Whether the file has a header line or not (a Boolean value)
mxPerc	A maximum threshold for the percentage the sample is allowed to have (defaults to 0.5)

Details

This function can be used to draw a random sample of lines from a very large CSV file. This is particularly usefull when you can not afford to load the file into memory to use R functions like `sample` to obtain the sample.

The function obtains the sample of rows without actually loading the full data into memory - only the final sample is loaded into main memory.

The function is based on unix-based utility programs (`perl` and `wc`) so it is limited to this type of platforms. The function will not run on other platforms (it will check the system variable `.Platform$OS.type`), although you may wish to check the function code and see if you can adapt it to your platform.

Value

A data frame

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Torgo, L. (2016) *Data Mining using R: learning with case studies, second edition*, Chapman & Hall/CRC (ISBN-13: 978-1482234893).

http://ltorgo.github.io/DMwR2

See Also

nrLinesFile, sample, sampleDBMS

`sampleDBMS` *Drawing a random sample of records of a table stored in a DBMS*

Description

Function for obtaining a random sample of records from a very large table stored in a databased managment system, whitout having to load in the full table into memory. Targets situations where the full data does not fit in the computer memory so usage of the standard `sample` function is not possible.

Usage

```
sampleDBMS(dbConn, tbl, percORn, mxPerc=0.5)
```

Arguments

`dbConn`	A data based connection object from the `DBI` package, that contains the result of establishing the connection to your target database in the respective database managment system.
`tbl`	A string containing the name of the (large) table in the database from which you want draw a random sample of records.
`percORn`	Either the percentage of number of rows of the file or the actual number of rows, the sample should have
`mxPerc`	A maximum threshold for the percentage the sample is allowed to have (defaults to 0.5)

Details

This function can be used to draw a random sample of records from a very large table of a database managment system. This is particularly usefull when you can not afford to load the full table into memory to use R functions like `sample` to obtain the sample.

The function obtains the sample of rows without actually loading the full data into memory - only the final sample is loaded into main memory.

The function assumes you have alread established and opened a connection to the database and receives as argument the DBI connection object.

Value

A data frame

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Torgo, L. (2016) *Data Mining using R: learning with case studies, second edition*, Chapman & Hall/CRC (ISBN-13: 978-1482234893).

http://ltorgo.github.io/DMwR2

See Also

sampleCSV, sample

Examples

```
## A simple example over a table on a MySQL database
## Not run:
library(DBI)
library(RMySQL)
drv <- dbDriver("MySQL")  # Loading the MySQL driver
con <- dbConnect(drv,dbname="myDB",
                 username="myUSER",password="myPASS",
                 host="localhost")
d <- sampleDBMS(con,"largeTable",10000)

## End(Not run)
```

SelfTrain *Self train a model on semi-supervised data*

Description

This function can be used to learn a classification model from semi-supervised data. This type of data includes observations for which the class label is known as well as observation with unknown class. The function implements a strategy known as self-training to be able to cope with this semi-supervised learning problem. The function can be applied to any classification algorithm that is able to obtain class probabilities when asked to classify a set of test cases (see the Details section).

Usage

```
SelfTrain(form,data,
          learner, learner.pars=list(),
          pred, pred.pars=list(),
          thrConf=0.9,
          maxIts=10,percFull=1,
          verbose=FALSE)
```

Arguments

form	A formula describing the prediction problem.
data	A data frame containing the available training set that is supposed to contain some rows for which the value of the target variable is unknown (i.e. equal to NA).
learner	An object of class learner (see class?learner for details), indicating the base classification algorithm to use in the self-training process.
learner.pars	A list with parameters that are to be passed to the learner function at each self-training iteration.
pred	A string with the name of a function that will carry out the probabilistic classification tasks that will be necessary during the self training process (see the Details section).
pred.pars	A list with parameters that are to be passed to the pred function at each self-training iteration when obtaining the predictions of the models.
thrConf	A number between 0 and 1, indicating the required classification confidence for an unlabelled case to be added to the labelled data set with the label predicted predicted by the classification algorithm.
maxIts	The maximum number of iterations of the self-training process.
percFull	A number between 0 and 1. If the percentage of labelled cases reaches this value the self-training process is stoped.
verbose	A boolean indicating the verbosity level of the function.

Details

Self-training (e.g. Yarowsky, 1995) is a well-known strategy to handle classification problems where a subset of the available training data has an unknown class label. The general idea is to use an iterative process where at each step we try to augment the set of labelled cases by "asking" the current classification model to label the unlabelled cases and choosing the ones for which the model is more confident on the assigned label to be added to the labeled set. With this extended set of labelled cases a new classification model is learned and the process is repeated until certain termination criteria are met.

This implementation of the self-training algorithm is generic in the sense that it can be used with any baseline classification learner provided this model is able to produce confidence scores for its predictions. The user needs to take care of the learning of the models and of the classification of the unlabelled cases. This is done as follows. The user supplies a learner object (see class?learner for details) in parameter learner to represent the algorithm to be used to obtain the classification models on each iteration of the self-training process. Furthermore, the user should create a function, whose named should be given in the parameter predFunc, that takes care of the classification of the currently unlabelled cases, on each iteration. This function should be written so that it receives as first argument the learned classification model (with the current training set), and a data frame with test cases in the second argument. This user-defined function should return a data frame with two columns and as many rows as there are rows in the given test set. The first column of this data frame should contain the assigned class labels by the provided classification model, for the respective test case. The second column should contain the confidence (a number between 0 and 1) associated to that classification. See the Examples section for an illustration of such user-defined function.

This function implements the iterative process of self training. On each iteration the provided learner is called with the set of labelled cases within the given data set. Unlabelled cases should have the value NA on the column of the target variable. The obtained classification model is then passed to the user-defined "predFunc" function together with the subset of the data that is unlabelled. As mentioned above this function returns a set of predicted class labels and the respective confidence. Test cases with confidence above the user-specified threshold (parameter thrConf) will be added to the labelled training set, with the label assigned by the current model. With this new training set a new classification model is obtained and the overall process repeated.

The self-training process stops if either there are no classifications that reach the required confidence level, if the maximum number of iterations is reached, or if the size of the current labelled training set is alread the target percentage of the given data set.

Value

This function returns a classification model. This will be an object of the same class as the object returned by the base classification learned provided by the user.

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Torgo, L. (2016) *Data Mining using R: learning with case studies, second edition*, Chapman & Hall/CRC (ISBN-13: 978-1482234893).

http://ltorgo.github.io/DMwR2

Yarowski, D. (1995). *Unsupervised word sense disambiguation rivaling supervised methods*. In Proceedings of the 33rd Annual Meeting of the association for Computational Linguistics (ACL), pages 189-196.

Examples

```
## Small example with the Iris classification data set
data(iris)

## Dividing the data set into train and test sets
idx <- sample(150,100)
tr <- iris[idx,]
ts <- iris[-idx,]

## Learn a tree with the full train set and test it
stdTree <- rpartXse(Species~ .,tr,se=0.5)
table(predict(stdTree,ts,type='class'),ts$Species)

## Now let us create another training set with most of the target
## variable values unknown
trSelfT <- tr
nas <- sample(100,70)
trSelfT[nas,'Species'] <- NA
```

```
## Learn a tree using only the labelled cases and test it
baseTree <- rpartXse(Species~ .,trSelfT[-nas,],se=0.5)
table(predict(baseTree,ts,type='class'),ts$Species)

## The user-defined function that will be used in the self-training process
f <- function(m,d) {
      l <- predict(m,d,type='class')
      c <- apply(predict(m,d),1,max)
      data.frame(cl=l,p=c)
}

## Self train the same model using the semi-superside data and test the
## resulting model
treeSelfT <- SelfTrain(Species~ .,trSelfT,'rpartXse',list(se=0.5),'f')
table(predict(treeSelfT,ts,type='class'),ts$Species)
```

`sigs.PR` *Precision and recall of a set of predicted trading signals*

Description

This function calculates the values of Precision and Recall of a set of predicted signals, given the set of true signals. The function assumes three types of signals: 'b' (Buy), 's' (Sell) and 'h' (Hold). The function returns the values of Precision and Recall for the buy, sell and sell+buy signals.

Usage

```
sigs.PR(preds, trues)
```

Arguments

`preds`	A factor with the predicted signals (values should be 'b','s', or 'h')
`trues`	A factor with the predicted signals (values should be 'b','s', or 'h')

Details

Precision and recall are two evaluation statistics often used to evaluate predictions for rare events. In this case we are talking about buy and sell opportunities.

Precision is the proportion of the events signaled by a model that actually occurred. Recall is a proportion of events that occurred that the model was able to capture. Ideally, the models should aim to obtain 100% precision and recall. However, it is often the case that there is a trade-off between the two statistics.

Value

A matrix with three rows and two columns. The columns are the values of Precision and Recall, respectively. The rows are the values for the three different events (sell, buy and sell+buy).

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Torgo, L. (2016) *Data Mining using R: learning with case studies, second edition*, Chapman & Hall/CRC (ISBN-13: 978-1482234893).

http://ltorgo.github.io/DMwR2

See Also

trading.signals, tradingEvaluation, trading.simulator

Examples

```
## A simple illustrative example use with random signals
ind <- rnorm(sd=0.3,100)
sigs <- trading.signals(ind,b.t=0.1,s.t=-0.1)
indT <- rnorm(sd=0.3,100)
sigsT <- trading.signals(indT,b.t=0.1,s.t=-0.1)
sigs.PR(sigs,sigsT)
```

SoftMax *Normalize a set of continuous values using SoftMax*

Description

Function for normalizing the range of values of a continuous variable using the SoftMax function (Pyle, 199).

Usage

```
SoftMax(x, lambda = 2, avg = mean(x, na.rm = T), std = sd(x, na.rm = T))
```

Arguments

x	A vector with numeric values
lambda	A numeric value entering the formula of the soft max function (see Details). Defaults to 2.
avg	The statistic of centrality of the continuous variable being normalized (defaults to the mean of the values in x).
std	The statistic of spread of the continuous variable being normalized (defaults to the standard deviation of the values in x).

Details

The Soft Max normalization consist in transforming the value x into

1 / [1+ exp((x-AVG(x))/(LAMBDA*SD(X)/2*PI))]

Value

An object with the same dimensions as x but with the values normalized

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Pyle, D. (1999). *Data preparation for data mining*. Morgan Kaufmann.

Torgo, L. (2016) *Data Mining using R: learning with case studies, second edition*, Chapman & Hall/CRC (ISBN-13: 978-1482234893).

http://ltorgo.github.io/DMwR2

See Also

scale

Examples

```
## A simple example with the iris data set
data(iris)
summary(SoftMax(iris[["Petal.Length"]]))
summary(iris[["Petal.Length"]])
```

---

sp500 *A set of daily quotes for SP500 in CSV Format*

Description

This is a CSV file containing the daily quotes of the SP500 sotck index from 1970-01-02 till 2015-12-25 . For each day information is given on the Open, High, Low and Close prices, and also for the Volume and Adjusted close price.

Usage

```
sp500
```

Format

A CSV file with a data matrix.

Source

Yahoo Finance

`test.algae` *Testing data for predicting algae blooms*

Description

This data set contains observations on 11 variables as well as the concentration levels of 7 harmful algae. Values were measured in several European rivers. The 11 predictor variables include 3 contextual variables (season, size and speed) describing the water sample, plus 8 chemical concentration measurements.

Usage

```
test.algae
```

Format

A data frame with 140 observations and 18 columns.

Source

ERUDIT http://www.erudit.de/ - European Network for Fuzzy Logic and Uncertainty Modelling in Information Technology.

`tradeRecord-class` *Class "tradeRecord"*

Description

This is a class that contains the result of a call to the function trading.simulator(). It contains information on the trading performance of a set of signals on a given set of "future" market quotes.

Objects from the Class

Objects can be created by calls of the form `tradeRecord(...)`. These objects contain information on i) the trading variables for each day in the simulation period; ii) on the positions hold during this period; iii) on the value used for transaction costs; iv) on the initial capital for the simulation; v) on the function that implements the trading policy used in the simulation; and vi) on the list of parameters of this function.

Slots

`trading`: Object of class `"xts"` containing the information on the trading activities through the testing period. This object has one line for each trading date. For each date it includes information on the closing price of the market ("Close"), on the order given at the end of that day ("Order"), on the money available to the trader at the end of that day ("Money"), on the number of stocks hold by the trader ("N.Stocks"), and on the equity at the end of that day ("Equity").

`positions`: Object of class `"matrix"` containing the positions hold by the trader during the simulation period. This is a matrix with seven columns, with as many rows as the number of positions hold by the trader. The columns of this matrix contain the type of position ("pos.type"), the number of stocks of the position ("N.stocks"), the date when the position was opened ("Odate"), the open price ("Oprice"), the closing date ("Cdate"), the closing price ("Cprice") and the percentage return of the position ("result").

`trans.cost`: Object of class `"numeric"` with the monetary value of each transaction (market order).

`init.cap`: Object of class `"numeric"` with the initial monetary value of the trader.

`policy.func`: Object of class `"character"` with the name of the function that should be called at the end of each day to decide what to do, i.e. the trading policy function. This function is called with the vector of signals till the current date, the market quotes till today, the current position of the trader and the currently available money.

`policy.pars`: Object of class `"list"` containing a list of extra parameters to be used when calling the trading policy function (these depend on the function defined by the user).

Methods

plot `signature(x = "tradeRecord", y = "ANY")`: provides a graphical representation of the trading results.

show `signature(object = "tradeRecord")`: shows an object in a proper way.

summary `signature(object = "tradeRecord")`: provides a summary of the trading results.

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Torgo, L. (2016) *Data Mining using R: learning with case studies, second edition*, Chapman & Hall/CRC (ISBN-13: 978-1482234893).

http://ltorgo.github.io/DMwR2

See Also

`trading.simulator`, `tradingEvaluation`

Examples

```
showClass("tradeRecord")
```

`trading.signals` *Discretize a set of values into a set of trading signals*

Description

This function transforms a set of numeric values into a set of trading signals according to two thresholds: one that establishes the limit above which any value will be transformed into a buy signal ('b'), and the other that sets the value below which we have a sell signal ('s'). Between the two thresholds we will have a hold signal ('h').

Usage

```
trading.signals(vs, b.t, s.t)
```

Arguments

`vs`	A vector with numeric values
`b.t`	A number representing the buy threshold
`s.t`	A number representing the sell threshold

Value

A factor with three possible values 'b' (buy), 's' (sell) or 'h' (hold)

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Torgo, L. (2016) *Data Mining using R: learning with case studies, second edition*, Chapman & Hall/CRC (ISBN-13: 978-1482234893).

http://ltorgo.github.io/DMwR2

See Also

`trading.signals`, `tradingEvaluation`, `trading.simulator`

Examples

```
trading.signals(rnorm(sd=0.5,100),b.t=0.1,s.t=-0.12)
```

`trading.simulator` *Simulate daily trading using a set of trading signals*

Description

This function can be used to obtain the trading performance of a set of signals by simulating daily trading on a market with these signals according to a user-defined trading policy. The idea is that the user supplies the actual quotes for the simulation period together with the trading signals to use during this period. On top of that the user also supplies a function implementing the trading policy to use. The result is a trading record for this period. This result can then be inspected and used to obtain several trading performance metrics with other functions.

Usage

```
trading.simulator(market, signals,
                  policy.func, policy.pars = list(),
                  trans.cost = 5, init.cap = 1e+06)
```

Arguments

`market`	A `xts` object containing the market quotes for each day of the simulation period. This object should contain at least the Open, High, Low and Close quotes for each day. These quotes (with these exact names) are used within the function and thus are required.
`signals`	A factor with as many signals as there are rows in the `market` `xts` object, i.e. as many signals as there are trading days in the simulation period. The signals should be 'b' for Buy, 's' for Sell and 'h' for Hold (actually this information is solely processed within the user-defined trading policy function which means that the values may be whatever the writer of this function wants).
`policy.func`	A string with the name of the function that will be called at the end of each day of the trading period. This user-defined function implements the trading policy to be used in the simulation. See the Details section for understanding what is the task of this function.
`policy.pars`	A list with parameters that are passed to the user-defined trading policy function when it is called at the end of each day.
`trans.cost`	A number with the cost of each market transaction (defaults to 5 monetary units).
`init.cap`	A number with the initial amount of money available for trading at the start of the simulation period (defaults to 1,000,000 monetary units).

Details

This function can be used to simulate daily trading according to a set of signals. The main parameters of this function are the market quotes for the simulation period and the model signals for this period. Two other parameters are the name of the user-defined trading policy function and its list of parameters. Finally, we can also specify the cost of each transaction and the initial capital available for the trader. The simulator will call the user-provided trading policy function at the end of each

daily section, and the function should return the orders that it wants the simulator to carry out. The simulator carries out these orders on the market and records all activity on several data structures. The result of the simulator is an object of class `tradeRecord` containing the information of this simulation. This object can then be used in other functions to obtain economic evaluation metrics or graphs of the trading activity.

The key issue in using this function is to create the user-defined trading policy function. These functions should be written using a certain protocol, that is, they should be aware of how the simulator will call them, and should return the information this simulator is expecting. At the end of each daily session d, the simulator calls the trading policy function with four main arguments plus any other parameters the user has provided in the call to the simulator in the parameter `policy.pars`. These four arguments are (1) a vector with the predicted signals until day d, (2) the market quotes (up to d), (3) the currently opened positions, and (4) the money currently available to the trader. The current positions are stored in a matrix with as many rows as there are open positions at the end of day d. This matrix has four columns: "pos.type" that can be 1 for a long position or -1 for a short position; "N.stocks", which is the number of stocks of the position; "Odate", which is the day on which the position was opened (a number between 1 and d); and "Oprice", which is the price at which the position was opened. The row names of this matrix contain the IDs of the positions that are relevant when we want to indicate the simulator that a certain position is to be closed. All this information is provided by the simulator to ensure the user can define a broad set of trading policy functions. The user-defined functions should return a data frame with a set of orders that the simulator should carry out. This data frame should include the following information (columns): "order", which should be 1 for buy orders and -1 for sell orders; "order.type", which should be 1 for market orders that are to be carried out immediately (actually at next day open price), 2 for limit orders or 3 for stop orders; "val", which should be the quantity of stocks to trade for opening market orders, NA for closing market orders, or a target price for limit and stop orders; "action", which should be "open" for orders that are opening a new position or "close" for orders closing an existing position; and finally, "posID", which should contain the ID of the position that is being closed, if applicable.

Value

An object of class `tradeRecord` (see 'class?tradeRecord' for details).

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Torgo, L. (2016) *Data Mining using R: learning with case studies, second edition*, Chapman & Hall/CRC (ISBN-13: 978-1482234893).

http://ltorgo.github.io/DMwR2

See Also

`tradingEvaluation`, `tradeRecord`, `trading.signals`, `sigs.PR`

Examples

```
## An example partially taken from chapter 3 of my book Data Mining
## with R (Torgo,2010)

## First a trading policy function
## This function implements a strategy to trade on futures with
## long and short positions. Its main ideas are the following:
## - all decisions aretaken at the end of the day, that is, after
## knowing all daily quotes of the current session.
## - if at the end of day d our models issue a sell signal and  we
## currently  do not hold any opened position, we will open a short
## position  by issuing a sell order. When this order is carried out  by
## the market at a price  pr sometime in the  future, we will
## immediately post two other orders. The first is a buy limit order
## with  a limit price of pr - p%, where p% is a target profit margin.
## We will wait 10 days for this target to be reached. If the  order  is
## not carried out by this deadline, we will buy at the closing price
## of  the 10th day. The second order is a buy stop order with a  price
## limit  pr + l%. This order is placed with the goal of limiting our
## eventual  losses with this position. The order will be executed if
## the  market reaches the price pr + l%, thus limiting our possible
## losses  to l%.
## - if the end of the day signal is buy the strategy is more or less
## the inverse
## Not run:
library(xts)
policy.1 <- function(signals,market,opened.pos,money,
                     bet=0.2,hold.time=10,
                     exp.prof=0.025, max.loss= 0.05
                     )
  {
    d <- NROW(market) # this is the ID of today
    orders <- NULL
    nOs <- NROW(opened.pos)
    # nothing to do!
    if (!nOs && signals[d] == 'h') return(orders)

    # First lets check if we can open new positions
    # i) long positions
    if (signals[d] == 'b' && !nOs) {
      quant <- round(bet*money/market[d,'Close'],0)
      if (quant > 0)
        orders <- rbind(orders,
              data.frame(order=c(1,-1,-1),order.type=c(1,2,3),
                         val = c(quant,
                                 market[d,'Close']*(1+exp.prof),
                                 market[d,'Close']*(1-max.loss)
                                ),
                         action = c('open','close','close'),
                         posID = c(NA,NA,NA)
                        )
                       )
```

```
  # ii) short positions
  } else if (signals[d] == 's' && !nOs) {
    # this is the nr of stocks we already need to buy
    # because of currently opened short positions
    need2buy <- sum(opened.pos[opened.pos[,'pos.type']==-1,
                               "N.stocks"])*market[d,'Close']
    quant <- round(bet*(money-need2buy)/market[d,'Close'],0)
    if (quant > 0)
      orders <- rbind(orders,
            data.frame(order=c(-1,1,1),order.type=c(1,2,3),
                       val = c(quant,
                               market[d,'Close']*(1-exp.prof),
                               market[d,'Close']*(1+max.loss)
                              ),
                       action = c('open','close','close'),
                       posID = c(NA,NA,NA)
                      )
                     )
  }

  # Now lets check if we need to close positions
  # because their holding time is over
  if (nOs)
    for(i in 1:nOs) {
      if (d - opened.pos[i,'Odate'] >= hold.time)
        orders <- rbind(orders,
              data.frame(order=-opened.pos[i,'pos.type'],
                         order.type=1,
                         val = NA,
                         action = 'close',
                         posID = rownames(opened.pos)[i]
                        )
                        )
    }

  orders
}

## Now let us play a bit with the SP500 quotes availabe in our package
data(GSPC)

## Let us select the last 3 months as the simulation period
market <- last(GSPC,'3 months')

## now let us generate a set of random trading signals for
## illustration purpose only
ndays <- nrow(market)
aRandomIndicator <- rnorm(sd=0.3,ndays)
theRandomSignals <- trading.signals(aRandomIndicator,b.t=0.1,s.t=-0.1)

## now lets trade!
tradeR <- trading.simulator(market,theRandomSignals,
```

```
                'policy.1',list(exp.prof=0.05,bet=0.2,hold.time=10))

  ## a few stats on the trading performance
  summary(tradeR)
  tradingEvaluation(tradeR)

## End(Not run)
  ## See the performance graphically
  ## Not run:
    plot(tradeR,market)

## End(Not run)
```

`tradingEvaluation` *Obtain a set of evaluation metrics for a set of trading actions*

Description

This function receives as argument an object of class `tradeRecord` that is the result of a call to the `trading.simulator()` function and produces a set of evaluation metrics of this simulation

Usage

```
tradingEvaluation(t)
```

Arguments

`t`	An object of call `tradeRecord` (see 'class?tradeRecord' for details)

Details

Given the result of a trading simulation this function calculates:

- The number of trades.
- The number of profitable trades.
- The percentage of profitable trades.
- The profit/loss of the simulation (i.e. the final result).
- The return of the simulation.
- The return over the buy and hold strategy.
- The maximum draw down of the simulation.
- The Sharpe Ration score.
- The average percentage return of the profitable trades.
- The average percentage return of the non-profitable trades.
- The average percentage return of all trades.
- The maximum return of all trades.
- The maximum percentage loss of all trades.

Value

A vector of evaluation metric values

Author(s)

Luis Torgo <ltorgo@dcc.fc.up.pt>

References

Torgo, L. (2016) *Data Mining using R: learning with case studies, second edition*, Chapman & Hall/CRC (ISBN-13: 978-1482234893).

http://ltorgo.github.io/DMwR2

See Also

tradeRecord, trading.simulator, trading.signals

Examples

```
## An example partially taken from chapter 3 of my book Data Mining
## with R (Torgo,2010)

## First a trading policy function
## This function implements a strategy to trade on futures with
## long and short positions. Its main ideas are the following:
## - all decisions aretaken at the end of the day, that is, after
## knowing all daily quotes of the current session.
## - if at the end of day d our models issue a sell signal and  we
## currently  do not hold any opened position, we will open a short
## position  by issuing a sell order. When this order is carried out  by
## the market at a price  pr sometime in the  future, we will
## immediately post two other orders. The first is a buy limit order
## with  a limit price of pr - p%, where p% is a target profit margin.
## We will wait 10 days for this target to be reached. If the  order  is
## not carried out by this deadline, we will buy at the closing price
## of  the 10th day. The second order is a buy stop order with a  price
## limit  pr + l%. This order is placed with the goal of limiting our
## eventual  losses with this position. The order will be executed if
## the  market reaches the price pr + l%, thus limiting our possible
## losses  to l%.
## - if the end of the day signal is buy the strategy is more or less
## the inverse
## Not run:
library(xts)
policy.1 <- function(signals,market,opened.pos,money,
                     bet=0.2,hold.time=10,
                     exp.prof=0.025, max.loss= 0.05
                     )
  {
    d <- NROW(market) # this is the ID of today
    orders <- NULL
```

```
nOs <- NROW(opened.pos)
# nothing to do!
if (!nOs && signals[d] == 'h') return(orders)

# First lets check if we can open new positions
# i) long positions
if (signals[d] == 'b' && !nOs) {
  quant <- round(bet*money/market[d,'Close'],0)
  if (quant > 0)
    orders <- rbind(orders,
          data.frame(order=c(1,-1,-1),order.type=c(1,2,3),
                     val = c(quant,
                             market[d,'Close']*(1+exp.prof),
                             market[d,'Close']*(1-max.loss)
                            ),
                     action = c('open','close','close'),
                     posID = c(NA,NA,NA)
                    )
                   )

# ii) short positions
} else if (signals[d] == 's' && !nOs) {
  # this is the nr of stocks we already need to buy
  # because of currently opened short positions
  need2buy <- sum(opened.pos[opened.pos[,'pos.type']==-1,
                             "N.stocks"])*market[d,'Close']
  quant <- round(bet*(money-need2buy)/market[d,'Close'],0)
  if (quant > 0)
    orders <- rbind(orders,
          data.frame(order=c(-1,1,1),order.type=c(1,2,3),
                     val = c(quant,
                             market[d,'Close']*(1-exp.prof),
                             market[d,'Close']*(1+max.loss)
                            ),
                     action = c('open','close','close'),
                     posID = c(NA,NA,NA)
                    )
                   )
}

# Now lets check if we need to close positions
# because their holding time is over
if (nOs)
  for(i in 1:nOs) {
    if (d - opened.pos[i,'Odate'] >= hold.time)
      orders <- rbind(orders,
            data.frame(order=-opened.pos[i,'pos.type'],
                       order.type=1,
                       val = NA,
                       action = 'close',
                       posID = rownames(opened.pos)[i]
                      )
                     )
```

```
    }

  orders
}

## Now let us play a bit with the SP500 quotes availabe in our package
data(GSPC)

## Let us select the last 3 months as the simulation period
market <- last(GSPC,'3 months')

## now let us generate a set of random trading signals for
## illustration purpose only
ndays <- nrow(market)
aRandomIndicator <- rnorm(sd=0.3,ndays)
theRandomSignals <- trading.signals(aRandomIndicator,b.t=0.1,s.t=-0.1)

## now lets trade!
tradeR <- trading.simulator(market,theRandomSignals,
            'policy.1',list(exp.prof=0.05,bet=0.2,hold.time=10))

## a few stats on the trading performance
tradingEvaluation(tradeR)

## End(Not run)
```

Index