Package ‘RFmarkerDetector’

February 29, 2016

Type Package

Title Multivariate Analysis of Metabolomics Data using Random Forests

Version 1.0.1

Date 2016-02-28

Author Piergiorgio Palla, Giuliano Armano

Maintainer Piergiorgio Palla <piergiorgio.palla@diee.unica.it>

Description A collection of tools for multivariate analysis of metabolomics data, which includes several preprocessing methods (normalization, scaling) and various exploration and data visualization techniques (Principal Components Analysis and Multi Dimensional Scaling). The core of the package is the Random Forest algorithm used for the construction, optimization and validation of classification models with the aim of identifying potentially relevant biomarkers.

License GPL-3

LazyData true

Imports randomForest, ggplot2, UsingR, WilcoxCV, ROCR, methods

Depends AUCRF

RoxygenNote 5.0.1

NeedsCompilation no

Repository CRAN

Date/Publication 2016-02-29 01:29:57

R topics documented:

`aucMCV` *AUC multiple cross-validation*

Description

This function implements the AUCRF algorithm for identifying the variables (metabolites) most relevant for the classification task

Usage

```
aucMCV(data, seed = 1234, ref_level = levels(data[, 2])[1],
  auc_rank = "MDG", auc_ntree = 500, auc_nfolds = 5, auc_pdel = 0.2,
  auc_colour = "grey", auc_iterations = 5)
```

Arguments

`data`	a n x p dataframe used to execute the AUCRF algorithm and perform a repetead CV of the AUCRF process. The dependent variable must be a binary variable defined as a `factor` and codified as 0 for negatives (e.g controls) and 1 for positivies (e.g. cases)
`seed`	a numeric value to set the seed of R's random number generator
`ref_level`	the class assumed as reference for the binary classification

`auc_rank`	the importance measure provided by `randomForest` for ranking the variables. There are two options: MDG (default) and MDA
`auc_ntree`	the number of tree of each random forest model used
`auc_nfolds`	the number of folds in cross validation. By default a 5-fold cross validation is performed
`auc_pdel`	the fraction of variables to be removed at each step. If $auc_pdel = 0$, it will be removed only one variable at each step
`auc_colour`	the color chosen
`auc_iterations`	a numeric that represents the number of cross validation repetitions

Details

Exploting the AUCRF algorithm, the fuction allows to identify the best performing 'parsimonious' model in terms of OOB-AUC and the most relevant variables (metabolites) involved in the prediction task.

Author(s)

Piergiorgio Palla

References

Calle ML, Urrea V, Boulesteix A-L, Malats N (2011) 'AUC-RF: A new strategy for genomic profiling with Random Forest'. Human Heredity

Examples

```
## data(cachexiaData)
## aucMCV(cachexiaData, ref_level = 'control')
```

`autoscale` *Unit variance scaling method performed on the columns of the data (i.e. metabolite concentrations measured by 1H NMR or binned 1H NMR spectra)*

Description

The function provides a data pretreatment approach called Autoscaling (also known as unit variance scaling). The data for each variable (metabolite) is mean centered and then divided by the standard deviation of the variable. This way each variable will have zero mean and unit standard deviation.

Usage

```
autoscale(data, exclude = T)
```

Arguments

`data`	a n x p data frame with n observations and p columns. While the first two columns usually represent the names of the samples and the class labels related to each sample respectively, the remaining columns represent metabolite concentrations measured by 1H NMR or bins of 1H NMR spectra
`exclude`	a boolean variable which stores a simple True/ False setting. If set to True the scaling method will exclude the first two columns.

Value

a scaled version of the input matrix

Author(s)

Piegiorgio Palla

Examples

```
## load the included example dataset
data(cachexiaData)
## call autoscale with the parameter exclude set to TRUE (default)
## in order to exclude the first two columns of the dataset from scaling
data.scaled <- autoscale(cachexiaData, exclude = TRUE)
```

`cachexiaData` *Metabolite concentrations*

Description

A dataset containing the metabolite concentrations of 77 urine samples from cancer patients measured by 1H NMR

Usage

```
data(cachexiaData)
```

Format

A data frame with 77 rows and 65 variables:

Source

http://dx.doi.org/10.1007/s11306-010-0232-9

`combinatorialRFMCCV` *Combinatorial Monte Carlo CV*

Description

This function performs a Monte Carlo CV for each of the Random Forest model grown considering all the k-combinations of the n input variables of the original dataset, with k ranging from 2 to n. It allows to get the most performing Random Forest model in terms of the AUC of the ROC curve and to obtain the most relevant input variables (metabolites or bins) associated with it.

Usage

```
combinatorialRFMCCV(dataset, parameters = list(ntrees = 500, nsplits = 100,
  test_prop = 1/3, kmax = 5))
```

Arguments

`dataset`	a n x p dataframe used to build the models. The first two columns must represent respectively the sample names and the class labels related to each sample
`parameters`	a list including the following parameters: • ntree the number of trees of each Random Forest model • nsplits the number of random splittings of the original dataset into training and test data sets • test_prop the percentage (expressed as a real number) of the observations of the original dataset • kmax the maximum number of inputs to combine.

Details

The function computes all the k-combinations of the n input variables, with k ranging from 2 to n. Each combination corresponds to a dataset on which the function will grow a Random Forest model, performing a Monte Carlo CV. Then it will provide the best performing model in terms of the AUC of the ROC curve and the most relevant variables associated with it.

Value

a list containing the most performing Random Forest model #' @examples ## data(cachexiaData) ## params <- list(ntrees = 100, nsplits = 10, test_prop = 1/3) ## res <- combinatorialRFMCCV(dataset = cachexiaData[,1:10], parameters = params) ## This task may take a long time depending on the ## dimension of the dataset and on the parameters provided

Author(s)

Piergiorgio Palla

`forestPerformance` *Characterizing the performance of a Random Forest model*

Description

This function provides the accuracy and the recall of a Random Forest model

Usage

```
forestPerformance(cm)
```

Arguments

`cm`	the confusion matrix of a validated Random Forest model

Value

a vector contining the accuracy, the recall and the same confusion matrix of the Random Forest model

Author(s)

Piergiorgio Palla

`getAvgAUC` *Computing the average AUC*

Description

This function computes the average AUC of the ROC curves obtained from a cross-validation procedure

Usage

```
getAvgAUC(predictions, labels)
```

Arguments

`predictions`	a vector, matrix, list, or data frame containing the predictions.
`labels`	a vector, matrix, list, or data frame containing the true class labels. It must have the same dimensions as predictions

Value

the average AUC value

Author(s)

Piergiorgio Palla

Examples

```
## load a simple dataset with the vectors of the predictions and the labels resulting from a CV
data(simpleData)
avg_auc <- getAvgAUC(simpleData$predictions, simpleData$labels)
```

`getBestRFModel` *Extracting the best performing Random Forest model*

Description

This function allows to find the best performing Random Forest model starting from a k-combination of its input variables

Usage

```
getBestRFModel(combinations, data, params)
```

Arguments

`combinations`	a k x n matrix in which n is the number of combinations of the input variables and k is the size of each combination
`data`	a n x p data frame of n observations and p-2 predictors. The first two columns must represent the sample names and the classes associates to each sample
`params`	a list of params useful to perform a Monte Carlo Cross validation. It should contain the following data: • ntrees the number of trees of each random forest model • nsplits the number of random splittings of the original dataset into training and test data sets • test_prop the percentage (expressed as a real number) of the observations of the original dataset to be included in each test set • ref_level the assumed reference class label

Details

The k-combinations of the input variables is represented as a k x n matrix in which k is the size of each combination and n is the number of combinations of the input variables of the original dataset. Each column of the combinations matrix contains the indexes of the input variables from the original dataset The `getBestRFModel` extracts a datAset from the original one considering the indexes in these columns. Then it will build a Random Forest model performing a Monte Carlo CV for each dataset. The models cross-validated will be compared considering the AUC of their averaged ROC curve. The function will return the best models, the maximum value of AUC and the most relevant input variables associated

Value

a list of the following elements:

- best_model_set the set of best performing Random Forest models in terms of AUC
- max_auc the maximum value of AUC corresponding to those models
- biomarker_set the set of metabolites (or bins) corresponding to the best performing Random Forest

Author(s)

Piergiorgio Palla

Examples

```
## data(cachexiaData)
## dataset <- cachexiaData[, 1:15]
## indexes <- 3:15
## combinations <- combn(x = indexes, m = 5) # a 5 x n_of_combinations matrix
## test_params = list(ntrees= 500, nsplits = 100, test_prop = 1/3)
## res <- getBestRFModel(combinations, dataset, test_params)
```

`lqvarFilter` *Filtering 'low quality' variables from the original dataset*

Description

This function takes the original dataset and filters those variables with a definite (usually relevant) percentage of zero-values

Usage

```
lqvarFilter(data, threshold = 0.5, exclude = T)
```

Arguments

`data`	a n x p data frame with n observations and p columns. While the first two columns usually represent the names of the samples and the class labels related to each sample respectively, the remaining columns represent metabolite concentrations measured by 1H NMR or bins of 1H NMR spectra
`threshold`	the percentage of zero values of a variable above which it will be eliminated from the dataset (default: 0.50)
`exclude`	a logical variable which stores a simple True / False setting. If set to True the filtering method will exclude the first two columns.

Value

a list containing the filtered dataset, a vector with the names of the varables excluded and a vector with the indexes of the variables eliminated

Author(s)

Piergiorgio Palla

Examples

```
 ## load the included example dataset
data(cachexiaData)
## call lqvarFilter with the parameter exclude set to TRUE (default)
## in order to exclude the first two columns of the dataset from scaling
res <- lqvarFilter(cachexiaData, threshold = 0.4, exclude = TRUE)
data.filtered <- res$filtered_dataset
```

---

mccv *mccv class*

Description

A constructor function for the S3 class mccv; the mccv class encapsulates information such as predictions and abels needed to plot roc curve(s) for a cross-validated random forest model

Usage

```
mccv(predictions, labels, models)
```

Arguments

`predictions`	a n x p dataframe of the predictions collected during the cross-validation process of the random forest, with n is equal to the number of samples in each test set and p is equal to the number of test sets
`labels`	a n x p dataframe of the labels of the samples included in each test set obtained splitting the original dataset during the cross-validation process of the model
`models`	a list of p random forest models tested in the cross-validation process

Value

an object of class mccv

Author(s)

Piergiorgio Palla

Examples

```
## load a simple dataset with the vectors of the predictions and the labels resulting from a CV
data(simpleData)
mccv_obj <- mccv(simpleData$predictions, simpleData$labels, models = NULL)
```

mds *mds class*

Description

A constructor function for the S3 class mds; the mds class encapsulates useful information for generating MDS plots

Usage

```
mds(dataset, opt = list(ntree = 1000, mtry = round(sqrt(ncol(dataset) - 2)),
  seed = 1234))
```

Arguments

`dataset`	a n x p dataframe representing the dataset to be used to train the model. The first two columns must represent respectively the samples names and the class labels related to each sample
`opt`	a list of optional parameters useful to train the random forest. It may include the number of trees (ntree), the parameter mtry and the seed

Value

an object of class mds including the following attributes:

- model an object of class random forest containing the proximity matrix
- classes a factor with the classes associated with each sample, used to train the model
- sample_names the vector of the names of the samples

Author(s)

Piergiorgio Palla

Examples

```
data(cachexiaData)
params = list(ntree = 1000, mtry = round(sqrt(ncol(cachexiaData) -2)), seed = 1234)
mds_obj <- mds(cachexiaData, opt = params)
```

`meanCenter` *Mean centering performed on the columns of the data (i.e. metabolite concentrations measured by 1H NMR or binned 1H NMR spectra)*

Description

The function allows to have each predictor (column) centered on zero. The average value of each predictor is substracted to each value in the column.

Usage

```
meanCenter(data, exclude = T)
```

Arguments

`data`	a n x p data frame with n observations and p columns. While the first two columns usually represent the names of the samples and the class labels related to each sample respectively, the remaining columns represent metabolite concentrations measured by 1H NMR or bins of 1H NMR spectra
`exclude`	a logical variable which stores a simple True / False setting. If set to True the scaling method will exclude the first two columns.

Value

a mean centered version of the input matrix

Author(s)

Piergiorgio Palla

Examples

```
## load the included example dataset
data(cachexiaData)
## call meanCenter with the parameter exclude set to TRUE (default)
## in order to exclude the first two columns of the dataset from scaling
data.scaled <- meanCenter(cachexiaData, exclude = TRUE)
```

`optimizeMTRY` *Mtry Optimization*

Description

This function provides a 'population' estimate of the average OOB error computed for different mtry values, starting from a sample of N models. These values will be used to compute the mtry associated to the minimum averaged OOB error, that is the optimal parameter we are looking for.

Usage

```
optimizeMTRY(oob_matrix)
```

Arguments

`oob_matrix`	a n x p of n OOB error values (one for each iteration) and p columns (one for each mtry value tested) Each value of a column is the oob error of a model growth with a particular mtry. Typically for each mtry, we will have N different models (N > 30), a sample large enough to provide an estimate of the average OOB error for the corresponding population of models.

Value

a list of two elements:

- mean_matrix a 1 x p matrix which contains the mean of each OOB errors sample (resulting from the training of N different Random Forest models growth with N different mtry values)
- ci_matrix a 2 x p matrix in which each column represents the 95% confidence interval of the mean of the population of the OOB errors for each mtry value
- sd_matrix a 1 x p matrix which contains the standard deviatiaon of each OOB error sample resulting from the training of N different models built for each value of mtry

Author(s)

Piergiorgio Palla

Examples

```
## data(cachexiaData)
## res <- tuneMTRY(cachexiaData, iterations = 50, maxntree = 600, mtry_length = 10, graph = F)
## l <- optimizeMTRY(res$oob)
```

`paretoscale`	*Pareto scaling method performed on the columns of the data table (i.e. metabolite concentrations measured by 1H NMR or binned 1H NMR spectra)*

Description

The function provides a data pretreatment approach called Pareto Scaling. Each column of the table is given a mean of zero by substracting the column column mean from each value in the column; then each value in each column is divided by a scaling factor, represented by the square root of the standard deviation of the column values.

Usage

```
paretoscale(data, exclude = T)
```

Arguments

`data`	a n x p matrix of n observations and p predictors. If the first two columns of the matrix represent respectively the sample names and the class labels associated to each sample, the scaling method should not include these two columns
`exclude`	a boolean variable. If set to True the scaling method will exclude the first two columns.

Details

This function is useful when variables have significantly different scales. It is generally the preferred option in NMR Metabolomics because it is a good compromise between no scaling (centering) and auto scaling

Value

a scaled version of the input matrix

Author(s)

Piegiorgio Palla

Examples

```
#' ## load the included example dataset
data(cachexiaData)
## call paretoscale with the parameter exclude set to TRUE (default)
## in order to exclude the first two columns of the dataset from scaling
data.scaled <- paretoscale(cachexiaData, exclude = TRUE)
```

pca *Principal Component Analysis*

Description

Performs a principal component analysis based on Singular Value Decomposition, on the given data matrix and returns the result as an object of the S3 class `pca`

Usage

```
pca(X, autoscale = T, exclude = T)
```

Arguments

`X`	a n x p data frame of n observations and p variables.
`autoscale`	a logical value indicating whether the variables should be autoscaled
`exclude`	a logical value indicating whether the first two columns should be excluded from the computation. The default is TRUE, because usually the first two columns of the dataset processed represent respectively the sample names and the class labels associated with the samples

Value

an S3 object of class `pca` with the following components:

- scores the scores matrix
- loadings the loading matrix
- variances the vector of variances explained by each PC
- classes the vector of the class labels associated with the samples
- features the vector with the names of the input variables

Author(s)

Piergiorgio Palla

Examples

```
data(cachexiaData)
pca_obj <- pca(cachexiaData, autoscale = TRUE, exclude = TRUE)
```

`plot.mccv` *Plotting single or multiple ROC curves of the cross-validated Random Forest models* `plot.mccv` *allows to plot single or multiple ROC curves to characterize the performace of a cross-validated Random Forest model*

Description

Plotting single or multiple ROC curves of the cross-validated Random Forest models `plot.mccv` allows to plot single or multiple ROC curves to characterize the performace of a cross-validated Random Forest model

Usage

```
## S3 method for class 'mccv'
plot(x, y, ..., opt = list(avg = "vertical", colorize = F))
```

Arguments

`x`	an object of class mccv
`y`	not used
`...`	optional graphical parameters
`opt`	a list containing the following optional parameters: • avg if the mccv object represents the predictions obtained from different cross-validation runs, we can have a different roc curve for each cv run. These curves can be averaged or not. Allowed values are none (plot all curves separately), horizontal (horizontal averaging), vertical(vertical averaging) and threshold (threshold averaging). • colorize a logical value which indicates if the curve(s) shoud be colorized according to the cutoff.

Author(s)

Piergiorgio Palla

Examples

```
## data(cachexiaData)
## params <- list(ntrees = 500, ref_level = levels(cachexiaData[,2])[1] )
## mccv_obj <- rfMCCV(cachexiaData, nsplits = 50, test_prop = 1/3, opt_params = params)
## params = list(avg = 'vertical', colorize = FALSE)
## plot.mccv(mccv_obj, opt = params)
```

`plot.mds` *Multi-dimensional Scaling (MDS) Plot*

Description

This function plots the scaling coordinates of the proximity matrix from random forest.

Usage

```
## S3 method for class 'mds'
plot(mds_obj, xrange, yrange)
```

Arguments

`mds_obj`	an object of class mds
`xrange`	a vector of two elements indicating the interval along the x-axis in which we want to display the names of the samples
`yrange`	a vector of two elements indicating the interval along the y-axis in which we want to display the names of the samples

Details

From the trained model we can get the dissimilarity matrix ** 1 - prox(i,j) ** The entries of this matrix can be seen as squared distances in a Euclidean high dimensional space. After having calculated scaling coordinates, we can project the data onto a lower dimensional space, preserving (as much as possible) the distances between the orginal points. This plot can be useful for discovering patterns in data.

Examples

```
## data(cachexiaData)
## params = list(ntree = 1000, mtry = round(sqrt(ncol(cachexiaData) -2)), seed = 1234)
## mds_obj <- mds(cachexiaData, opt = params)
## plot.mds(mds_obj = mds_obj)
```

`plot.pca.loadings` *PCA Loadings plot This function plots the relation between the original variables and the subspace dimensions. It is useful for interpreting relationships among variables.*

Description

PCA Loadings plot

This function plots the relation between the original variables and the subspace dimensions. It is useful for interpreting relationships among variables.

Usage

```
## S3 method for class 'pca.loadings'
plot(pca_obj, nvar)
```

Arguments

`pca_obj`	an object of class pca
`nvar`	the number of variables to plot
`...`	optional graphical parameters

Author(s)

Piergiorgio Palla

Examples

```
## data(cachexiaData)
## pca_obj <- pca(cachexiaData, autoscale = TRUE, exclude = TRUE)
## plot.pca.loadings(pca_obj, nvar = 20)
```

`plot.pca.scores` *PCA Scores plot This function creates a plot that graphically projects the original samples onto the subspce spanned by the first two principal components*

Description

PCA Scores plot

This function creates a plot that graphically projects the original samples onto the subspce spanned by the first two principal components

Usage

```
## S3 method for class 'pca.scores'
plot(pca_obj, dataset, xrange, yrange)
```

Arguments

`pca_obj`	an object of class pca
`dataset`	a n x p dataframe representing the dataset used to create the pca object
`xrange`	a vector of two elements indicating the range of values along the x-axis in which eventually it will be specified the name of the samples
`yrange`	a vector of two elements indicating the range of values along the y-axis in which eventually it will be specified the name of the samples

Details

The Scores plot is used for interpreting relations among the observations

Author(s)

Piergiorgio Palla

Examples

```
## data(cachexiaData)
## pca_obj <- pca(cachexiaData, autoscale = TRUE, exclude = TRUE)
## plot.pca.scores(pca_obj, cachexiaData)
```

`plotAUCvsCombinations` *Plotting the average AUC as a function of the number of combinations*

Description

This function allows to plot the average AUC as a function of the number k-combinations of the n input variables. If n is the number of input variables, the number of k-combinations of those variables is equal to $n!/k!(n! - k!)$. Each of these combinations contains the indexes of the input variables selected. For each combination we can extract a dataset, build a random forest model and perform a cross-validation. We can describe the performance of each cross-validated model with an 'average' ROC curve and its AUC. The collected auc values for each combination (dataset) are used by the function to build a diagram of the AUC as a function of the number of combinations

Usage

```
plotAUCvsCombinations(auc_values, num_of_variables, num_of_combinations)
```

Arguments

`auc_values`	an array with the auc values to plot
`num_of_variables`	the k dimension of each combination
`num_of_combinations`	the number of k combinations of the set of the input variables

Author(s)

Piergiorgio Palla

plotOOBvsMTRY — *Plotting the average OOB error and its 95% confidence interval as a function of the mtry parameter*

Description

Plotting the average OOB error and its 95% confidence interval as a function of the mtry parameter

Usage

```
plotOOBvsMTRY(mean_matrix, ci_matrix)
```

Arguments

`mean_matrix`	a 1 x p matrix where p is the number of mtry values tested. Each value represents the average OOB error obtained training multiple Random Forest models with a defined value of mtry
`ci_matrix`	a 2 x p matrix containing the extremes of the confidence interval of the average OOB error.

Value

a graphical representation of the average OOB error as a function of the mtry parameter.

Author(s)

Piergiorgio Palla

Examples

```
## data(cachexiaData)
## res <- tuneMTRY(cachexiaData, iterations = 5, maxntree = 600, mtry_length = 10, graph = F)
## l <- optimizeMTRY(res$oob)
## plotOOBvsMTRY(l$mean_matrix, l$ci_matrix)
```

plotVarFreq — *Variable Frequency Plot*

Description

This function plots the probability of selection of each variable (metabolite) as the proportion of times that is selected by AUCRF method

Usage

```
plotVarFreq(varFrequency, thd = 0.4, color = "blue", hcol = "red")
```

Arguments

`varFrequency`	a numeric vector with the probabilities of selection of each input variable
`thd`	a numeric value that indicates the lower limit of probability of variables to be represented. The default value is 0.4 (40%)
`color`	the color of the graph
`hcol`	the color of the horizontal line, indicating the lower limit of the probability of selection of variables

Author(s)

Piergiorgio Palla

Examples

```
## data(cachexiaData)
## rf.aucv1 <- aucMCV(cachexiaData, ref_level = 'control')
## plotVarFreq(varFrequency = rf.aucv1$Psel,  thd=0.55)
```

`rfMCCV` *Monte Carlo cross-validation of Random Forest models*

Description

This function allows to perform a Monte Carlo cross-validation of a Random Forest

Usage

```
rfMCCV(data, nsplits, test_prop, opt_params)
```

Arguments

`data`	a n x p dataframe used to build the models. The first two columns must represent respectively the sample names and the class labels related to each sample
`nsplits`	the number of random splittings of the original dataset into training and test data sets
`test_prop`	the percentage (expressed as a real number) of the observations of the original dataset to be included in each test set
`opt_params`	a list of optional parameters characterizing both the model to be validated and the input dataset. It may include parameters like the number of trees (ntree), the mtry, or the eventual reference class label (ref_level) of the dataset

Value

a list of three elements:

- a n x p dataframe representing the predictions during the cross-validation process. Its number of lines is equal to the number of observations included in each test set and the number of columns is equal to the number of test sets (defined by the nsplits input parameter).
- a n x p dataframe representing the labels associated to the samples of each test set. Its number of lines n is equal to the number observations in each test set while p is the number of test sets defined by the nsplits input parameter
- a list of p random forest models tested in the cross-validation process

@examples data(cachexiaData) params <- list(ntrees = 500, ref_level = levels(cachexiaData[,2])[1]) mccv_obj <- rfMCCV(cachexiaData, nsplits = 5, test_prop = 1/3, opt_params = params)

Author(s)

Piergiorgio Palla

`rfMCCVPerf` *Extracting average accuracy and recall of a list of Random Forest models*

Description

This function provides the average accuracy and the recall of a list of Random Forest models

Usage

```
rfMCCVPerf(model_list)
```

Arguments

`model_list`	a list of different Random Forest models

Value

a list of the two elements:

- avg_accuracy the average accuracy
- avg_recall the average recall

Author(s)

Piergiorgio Palla

Examples

```
## data(cachexiaData)
## params <- list(ntrees = 500, ref_level = levels(cachexiaData[,2])[1] )
## mccv_obj <- rfMCCV(cachexiaData, nsplits = 5, test_prop = 1/3, opt_params = params)
## models <- mccv_obj$models
## res <- rfMCCVPerf(models)
```

---

rsd *Computing relative standard deviation of a vector*

Description

This function computes the relative standard deviation (also known as coefficient of variation) of a numeric vector defined as the ratio of the standard deviation to the mean of the vector elements, expressed as percentage

Usage

```
rsd(v)
```

Arguments

v	a numeric vector

Details

the coefficient of variation shows the extent of variability in relation to mean of the population. It is expressed as a percentage. Lower values indicate lower variability.

Value

the value of the coefficient of variation of the input vector expressed as a percentage and rounded to two decimal places

Author(s)

Piergiorgio Palla

Examples

```
v <-  runif(10, min = 5, max = 30)
rsd(v)
```

rsdFilter *Filtering less informative variables*

Description

`rsdFilter` removes from the dataframe the predictors with a relative standard deviation less than or equal to an inserted threshold

Usage

```
rsdFilter(data, threshold, exclude = T)
```

Arguments

`data`	a n x p data frame with n observations and p columns. While the first two columns usually represent the names of the samples and the class labels related to each sample respectively, the remaining columns represent metabolite concentrations measured by 1H NMR or bins of 1H NMR spectra
`threshold`	a numeric value representing a limit: each predictor with a relative standard deviation lower than that will be removed form the dataframe
`exclude`	a logical variable which stores a simple True / False setting. If set to True the filtering method will exclude the first two columns.

Value

a list containing the filtered dataset, a vector with the names of the varables excluded and a vector with the indexes of the variables eliminated

Author(s)

Piergiorgio Palla

Examples

```
 ## load the included example dataset
data(cachexiaData)
## call rsdFilter with the parameter exclude set to TRUE (default)
## in order to exclude the first two columns of the dataset from scaling
data.filtered <- rsdFilter(cachexiaData, threshold = 15, exclude = TRUE)
```

screeplot *Scree Plot*

Description

This function creates a graphical display of the variances against the number of principal components. It is used to determine how many components should be retained in order to explain a high percentage of the variation in the data

Usage

```
screeplot(pca_obj, ncomp)
```

Arguments

pca_obj	an object of class pca
ncomp	the number of components to plot

Details

screplot generates 2 graphs that represent respectively the relative variances and the cumulative variances associated with the principal components

Author(s)

Piergiorgio Palla

Examples

```
## data(cachexiaData)
## pca_obj <- pca(cachexiaData, autoscale = TRUE, exclude = TRUE)
## screeplot(pca_obj, ncomp = 10)
```

simpleData *simpleData*

Description

A list of two vectors named respectively predictions and labels

Usage

```
data(simpleData)
```

Format

An object of class `list` of length 2.

`tuneMTRY` *Tuning of the mtry parameter for a Random Forest model*

Description

`tuneMTRY` tries to identify the 'optimal' value of the mtry parameter which indicates the number of input variables randomly chosen at each node

Usage

```
tuneMTRY(data, iterations, maxntree, mtry_length, changeTreeNum = F,
  graph = T)
```

Arguments

`data`	the n x p dataframe used to build the models and to tune the parameter mtry. The first two columns must represent respectively the sample names and the class labels related to each sample
`iterations`	the number of different random forest models built for each value of mtry
`maxntree`	the maximum number of trees of each random forest model
`mtry_length`	an integer value representing the number of mtry values to test.
`changeTreeNum`	a logical value indicating whether or not to change
`graph`	a logical value indicating whether to plot the OOB error as a function of the parameter mtry the number of trees during the tuning of mtry

Details

The function searches for the optimal value of mtry assigning to it a set of values and building different random forests (also with a different number of trees) for each value of the mtry. The number of models built for each mtry is defined by the `iteration` parameter. The oob errors of each random forest model, computed for each mtry value, are then arranged in a matrix

Value

a list of two elements:

- a diagram of the average value of the OOB error as a function of the mtry with its 95% confidence interval
- a n x p matrix of n iterations and p mtry values tested

Author(s)

Piergiorgio Palla

Examples

```
## data(cachexiaData)
## res <- tuneMTRY(cachexiaData, iterations = 10, maxntree = 600, mtry_length = 10, graph = FALSE)
```

tuneNTREE *Tuning of the ntree parameter (i.e. the number of trees) for a Random Forest model*

Description

This function tries to find the 'optimal' value for the parameter ntree which indicates the number of trees used to grow the ensemble of trees. To do that it will build several random forest models with a different number of trees for the mtry value considered. The number of models built for each ntree value will be equal to the parameter iteration. The oob errors of each random forest model, computed for each ntree value will be arranged in a matrix.

Usage

```
tuneNTREE(data, mtry, iterations, minNTREE = 500, pace = 100,
  seq_length = 5)
```

Arguments

`data`	the n x p dataframe used to build the Random Forest models. The first two columns must represent respectively the sample names and the class labels associated to each sample
`mtry`	the chosen mtry value
`iterations`	the number of Random Forest models to be built for each value of ntree
`minNTREE`	the minimum number of trees of each random forest model.
`pace`	the pace between each value of ntree to be tested
`seq_length`	the number of ntree values to be tested

Value

a n x p matrix in which n is the number of models considered and p is the number of ntree values tested. Each column represents the oob errors resulting from each model and corresponding to the different ntree values

Author(s)

Piergiorgio Palla

Examples

```
## data(cachexiaData)
## res <- tuneNTREE(cachexiaData, 8, iterations = 50, minNTREE = 600, pace = 100, seq_length = 10)
```

Index