Package 'specmine'

November 10, 2015

Type Package

Title Metabolomics and Spectral Data Analysis and Mining

Version 1.0

Date 2015-10-19

Author Christopher Costa <chrisbcl@hotmail.com>,
Marcelo Maraschin <mtocsy@gmail.com>,
Miguel Rocha <mrocha@di.uminho.pt>

Maintainer Christopher Costa <chrisbcl@hotmail.com>

Depends R (>= 3.1.0)

Imports compare, hyperSpec, ChemoSpec, baseline, rgl, Metrics, GGally, ggplot2, ellipse, ggdendro, caret, pls, pcaPP, RColorBrewer, grid, methods, qdap, MASS, scatterplot3d, xcms, MAIT, genefilter, impute

Description Provides a set of methods for metabolomics data analysis, including data loading in different formats, pre-processing, metabolite identification, univariate and multivariate data analysis, machine learning and feature selection. Case studies can be found on the website: http://darwin.di.uminho.pt/metabolomics .

License GPL (>= 2)

NeedsCompilation no

Repository CRAN

Date/Publication 2015-11-10 11:08:16

R topics documented:

`absorbance_to_transmittance`

Convert absorbance to transmittance

Description

Converts absorbance values to transmittance values.

Usage

```
absorbance_to_transmittance(dataset)
```

Arguments

`dataset`	list representing the dataset from a metabolomics experiment.

Value

Returns the dataset with the data points converted to transmittance values.

Examples

```
## Example of converting transmittance values to absorbance values
data(cassavaPPD)
cassavaPPD = transmittance_to_absorbance(cassavaPPD)
cassavaPPD = absorbance_to_transmittance(cassavaPPD)
```

---

aggregate_samples *Aggregate samples*

Description

Aggregate samples according to an aggregate function like mean, median, etc.

Usage

```
aggregate_samples(dataset, indexes, aggreg.fn = "mean",
meta.to.remove = c())
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
indexes	index vector with the samples that are going to be aggregated (e.g. c(1,1,2,2), this index vector will aggregate the first two samples and the last two samples).
aggreg.fn	aggregation function (e.g. "mean", "median", etc).
meta.to.remove	metadata's variables to be removed.

Value

Returns the dataset with the samples aggregated.

Examples

```
## Example of aggregating samples
data(cassavaPPD)
index.vector = c(1,1,2,2,2,2,2,1,1,3,3,3,3,3,1,4,4,4,4,4,5,5,6,6,6,6,6,5,5,
7,7,7,7,7,8,8,8,8,8,9,9,10,10,10,10,10,9,9,11,11,11,11,11,
9,12,12,12,12,12,13,13,14,14,14,14,14,13,13,15,15,15,15,15,
13,16,16,16,16,16)
dataset = aggregate_samples(cassavaPPD, index.vector, "mean")
```

aov_all_vars *Analysis of variance*

Description

Perform analysis of variance of all variables in the dataset.

Usage

```
aov_all_vars(dataset, column.class, doTukey = T, write.file = F,
file.out = "anova-res.csv")
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
column.class	string or index indicating what metadata to use.
doTukey	boolean value for do or do not TukeyHSD.
write.file	boolean value indicating if a file with the results is written or not.
file.out	name of the file if write.file is TRUE.

Value

Data frame with the results of ANOVA, with p-value, logarithm of p-value, false discovery rate (fdr) and tukey is doTukey is TRUE. The result is ordered by p-value.

Examples

```
  ## Example of ANOVA with TukeyHSD
  data(cassavaPPD)
  cassavaPPD = flat_pattern_filter(cassavaPPD, "iqr", by.percent = TRUE,
red.value = 75)
  result = aov_all_vars(cassavaPPD, "varieties", doTukey = FALSE)
```

apply_by_group *Apply by group*

Description

Apply a function to samples from a given metadata's group.

Usage

```
apply_by_group(dataset, fn.to.apply, metadata.var, var.value)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
fn.to.apply	function to apply (e.g. mean, max, min).
metadata.var	name of the metadata's variable.
var.value	value of the metadata's variable.

Value

Returns a vector with the variables and the value of the applied function.

Examples

```
## Example of applying a function to a group
data(cachexia)
apply.group.result = apply_by_group(cachexia, mean, "Muscle.loss",
"control")
```

---

apply_by_groups *Apply by groups*

Description

Apply a function to samples from a metadata's variable.

Usage

```
apply_by_groups(dataset, metadata.var, fn.to.apply = "mean",
variables = NULL, variable.bounds = NULL)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
metadata.var	name of the metadata's variable.
fn.to.apply	function to apply (e.g. mean, max, min).
variables	allows to define which variables to calculate the stats (if numbers, indexes are assumed).
variable.bounds	allow to define an interval of variables (if numeric).

Value

Returns a vector with the variables and the value of the applied function on the metadata's groups.

Examples

```
## Example of applying a function to groups
data(cachexia)
apply.groups.result = apply_by_groups(cachexia, "Muscle.loss", mean)
```

apply_by_sample *Apply function to samples*

Description

Applies a function to the values of each sample

Usage

```
apply_by_sample(dataset, fn.to.apply, samples = NULL, ...)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
fn.to.apply	function to apply (e.g. mean, max, min).
samples	if defined restricts the application to a given set of samples.
...	additional parameters to apply function.

Value

Returns a vector with the samples and the value of the applied function.

Examples

```
## Example of applying a function to variables
data(cachexia)
apply.samples.result = apply_by_sample(cachexia, mean)
```

apply_by_variable *Apply function to variables*

Description

Applies a function to the values of each variable

Usage

```
apply_by_variable(dataset, fn.to.apply, variables = NULL,
variable.bounds = NULL, samples = NULL, ...)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
fn.to.apply	function to apply (e.g. mean, max, min).
variables	allows to define which variables to calculate the stats (if numbers, indexes are assumed).
variable.bounds	allow to define an interval of variables (if numeric).
samples	if defined restricts the application to a given set of samples.
...	additional parameters to apply function.

Value

Returns a vector with the variables and the value of the applied function.

Examples

```
## Example of applying a function to variables
data(cachexia)
apply.variables.result = apply_by_variable(cachexia, mean)
```

---

background_correction *Background correction*

Description

Perform background correction on the spectra.

Usage

```
background_correction(dataset)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.

Value

Returns the dataset with background correction performed on the data.

Examples

```
## Example of background correction
library(hyperSpec)
data(flu)
flu.converted = convert_from_hyperspec(flu)
flu.corrected = background_correction(flu.converted)
```

`baseline_correction` *Baseline correction*

Description

Performs baseline correction on the dataset.

Usage

```
baseline_correction(dataset, method = "modpolyfit", ...)
```

Arguments

`dataset`	list representing the dataset from a metabolomics experiment.
`method`	string representing the baseline correction method. It can be one of these methods: • "als" Asymmetric Least Squares, baseline correction by 2nd derivative constrained weighted regression • "fillPeaks" An iterative algorithm using suppression of baseline by means in local windows • "irls" Iterative Restricted Least Squares, an algorithm with primary smoothing and repeated baseline suppressions and regressions with 2nd derivative constraint • "lowpass" Low-pass filter, an algorithm for removing baselines based on Fast Fourier Transform filtering • "medianWindow" an implementation and extension of Mark S. Friedrichs' model-free algorithm • "modpolyfit" Modified polynomial fitting, an implementation of Chad A. Lieber and Anita Mahadevan-Jansen's algorithm for polynomial fitting • "peakDetection" A translation from Kevin R. Coombes et al.'s MATLAB code for detecting peaks and removing baselines • "rfbaseline" Robust Baseline Estimation, Wrapper for Andreas F. Ruckstuhl, Matthew P. Jacobson, Robert W. Field, James A. Dodd's algorithm based on LOWESS and weighted regression • "rollingBall" Ideas from Rolling Ball algorithm for X-ray spectra by M. A. Kneen and H. J. Annegarn. Variable window width has been left out
`...`	Additional parameters of the baseline correction method.

Value

Returns the dataset with the data's baseline corrected.

Examples

```
## Example of baseline correction
data(cassavaPPD)
dataset.corrected = baseline_correction(cassavaPPD, method = "modpolyfit")
```

boxplot_variables *Boxplot of variables*

Description

Boxplot of each variable of the dataset.

Usage

```
boxplot_variables(dataset, variables = NULL, samples = NULL,
horizontal = T, col = "lightblue", nchar.label = 10,
cex.axis = 0.8, ...)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
variables	vector with the variables names or a NULL value indicating all variables.
samples	vector with the samples names or a NULL value indicating all samples.
horizontal	boolean value indicating if the boxplots should be horizontal.
col	string that represents the color of the bodies of the boxplots.
nchar.label	number of characters to display the variables' names.
cex.axis	numeric value that indicates the amount by which the axis is magnified relative to the default.
...	additional parameters of boxplot function.

Examples

```
## Example of showing the boxplot of a few variables
data(cachexia)
boxplot_variables(cachexia, variables = c("Creatine","Serine","Lactate"))
```

boxplot_vars_factor *Boxplot of variables with metadata's variable factors*

Description

Boxplot of variables with metadata's variable factors from the dataset.

Usage

```
boxplot_vars_factor(dataset, meta.var, variables = NULL,
samples = NULL, horizontal = F, nchar.label = 10, col = NULL,
vec.par = NULL, cex.axis = 0.8, ylabs = NULL, ...)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
meta.var	metadata's variable.
variables	vector with the variables names or a NULL value indicating all variables.
samples	vector with the samples names or a NULL value indicating all samples.
horizontal	boolean value indicating if the boxplots should be horizontal.
nchar.label	number of characters to display the variables' names.
col	string that represents the color of the bodies of the boxplots.
vec.par	vector with the disposition of the boxplots (rows, columns).
cex.axis	numeric value that indicates the amount by which the axis is magnified relative to the default.
ylabs	y-axis labels.
...	additional parameters of boxplot function.

Examples

```
## Example of showing the boxplot factors of a few variables
data(cachexia)
boxplot_vars_factor(cachexia, "Muscle.loss", variables = c("Creatine","Serine",
"Lactate"))
```

---

cachexia *Human Cachexia data*

Description

Cachexia is a complex metabolic syndrome associated with an underlying illness (such as cancer) and characterized by loss of muscle with or without loss of fat mass (Evans et al., 2008). A total of 77 urine samples were collected being 47 of them patients with cachexia, and 30 control patients.

Usage

```
data(cachexia)
```

Format

An object of class `"list"`

Source

MetaboAnalyst

References

Eisner et al. (2010) Learning to predict cancer-associated skeletal muscle wasting from 1h-nmr profiles of urinary metabolites Metabolomics 7:25-34

Examples

```
data(cachexia)
    sum_dataset(cachexia)
```

`cassavaPPD` *Cassava Postharvest Physiological Deterioration*

Description

Cassava is a root well known and widely cultivated in tropical and subtropical regions for its starchy tuberous root, which is a great source of carbohydrates. It also has a great variety of applications, like animal feeding, culinary or alcoholic beverages. In some countries, cassava has also been tested as an ethanol biofuel feedstock.

Usage

```
data(cassavaPPD)
```

Format

An object of class `"list"`

References

Uarrota et al. (2014) Metabolomics combined with chemometric tools (pca, hca, pls-da and svm) for screening cassava (manihot esculenta crantz) roots during postharvest physiological deterioration. Food Chemistry 161:67-78

Examples

```
data(cassavaPPD)
sum_dataset(cassavaPPD)
```

`check_dataset` *Check dataset*

Description

Check if the dataset is valid and if not give the proper error message.

Usage

```
check_dataset(dataset)
```

Arguments

`dataset`	list representing the dataset from a metabolomics experiment.

Value

Message saying if the dataset is valid or invalid, in the last case also gives the reason.

Examples

```
## Example checking a dataset
data(cachexia)
check_dataset(cachexia)
```

---

clustering *Perform cluster analysis*

Description

Perform cluster analysis on the dataset.

Usage

```
clustering(dataset, method = "hc", distance = "euclidean",
type = "samples", num.clusters = 5, clustMethod = "complete")
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
method	a string describing the method of clustering. Possible types are: • "hc" perform hierarchical clustering. • "kmeans" perform kmeans clustering.
distance	the distance measure to be used to compute the distances between the rows of a data matrix. Possible types are "euclidean", "manhattan", "pearson" or "spearman". Only for hierarchical clustering.
type	a string indicating if cluster analysis will be performed on samples ("samples") or on variables ("variables").
num.clusters	the number of clusters in k-means cluster analysis.
clustMethod	Cluster method for hierarchical clustering.

Value

An object of class kmeans or hclust with the clustering results.

Examples

```
## Example of kmeans and hierarchical clustering
data(cachexia)
hc.result = clustering(cachexia, method = "hc",
    distance = "euclidean")
kmeans.result = clustering(cachexia, method = "kmeans",
num.clusters = 4)
```

compare_regions_by_sample *Compare regions by sample*

Description

Compare two regions of a dataset by samples.

Usage

```
compare_regions_by_sample(dataset1, dataset2, fn.to.apply,
samples = NULL, ...)
```

Arguments

dataset1	list representing the dataset from a metabolomics experiment.
dataset2	list representing the dataset from a metabolomics experiment.
fn.to.apply	function to apply (e.g. mean, max, min).
samples	if defined restricts the application to a given set of samples.
...	additional parameters to apply.by.sample function.

Value

Returns a data.frame with the results of the function applied to the samples and the ration between the two datasets.

Examples

```
## Example of comparing regions by sample
data(cachexia)
subset1 = subset_x_values(cachexia, 1:31, by.index = TRUE)
subset2 = subset_x_values(cachexia, 32:63, by.index = TRUE)
comp.regions.result = compare_regions_by_sample(subset1, subset2,
mean)
```

convert_from_chemospec *Convert from ChemoSpec*

Description

Convert the dataset in the ChemoSpec format to a dataset of this package.

Usage

```
convert_from_chemospec(csobj, type = "undefined",
description = "")
```

Arguments

csobj	ChemoSpec object representing the dataset.
type	string representing the type of the data.
description	string representing the description of the dataset.

Value

Returns a list representing the dataset converted.

Examples

```
## Not run:
  ## Example of converting a dataset from ChemoSpec
  dataset = convert_from_chemospec(chemospec.dataset, type = "nmr-peaks",
        description = "some description")

## End(Not run)
```

---

convert_from_hyperspec

Convert from hyperspec

Description

Convert the dataset in the hyperspec format to a dataset of this package.

Usage

```
convert_from_hyperspec(hsobj, type = "undefined",
description = "")
```

Arguments

hsobj	hyperspec object representing the dataset.
type	string representing the type of the data.
description	string representing the description of the dataset.

Value

Returns a list representing the dataset converted.

Examples

```
## Example of converting a dataset from hyperspec
library(hyperSpec)
data(flu)
dataset = convert_from_hyperspec(flu, type = "undefined",
  description = "some description")
```

`convert_to_factor` *Convert metadata to factor*

Description

Convert a metadata's variable to factor.

Usage

```
convert_to_factor(dataset, metadata.var)
```

Arguments

`dataset`	list representing the dataset from a metabolomics experiment.
`metadata.var`	name of the metadata's variable.

Value

Returns the dataset with the metadata's variable converted to factor.

Examples

```
## Example of converting a metadata's variable to factor
data(cassavaPPD)
dataset = convert_to_factor(cassavaPPD, "ppds")
```

`convert_to_hyperspec` *Convert to hyperspec*

Description

Convert a dataset to an hyperspec object.

Usage

```
convert_to_hyperspec(dataset)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.

Value

Returns an hyperspec object representing the dataset converted.

Examples

```
## Example of converting a dataset to an hyperspec object
data(cassavaPPD)
hyperspec.cassava = convert_to_hyperspec(cassavaPPD)
```

---

correlations_dataset *Dataset correlations*

Description

Calculate the correlations of all variables or samples in the dataset.

Usage

```
correlations_dataset(dataset, method = "pearson", by.var = T)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
method	correlation method, it can be "pearson", "kendall" or "spearman".
by.var	if TRUE then the correlations of the variables will be calculated, if not then the correlations of the samples will be calculated.

Value

Returns the correlation matrix

Examples

```
## Example of correlations of variables
data(cachexia)
corr.result = correlations_dataset(cachexia,
method = "pearson", by.var = TRUE)
```

correlations_test *Correlations test*

Description

Performs correlations test to the whole dataset.

Usage

```
correlations_test(dataset, method = "pearson", by.var = T,
alternative = "two.sided")
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
method	correlation method, it can be "pearson", "kendall" or "spearman".
by.var	if TRUE then the correlations of the variables will be calculated, if not then the correlations of the samples will be calculated.
alternative	alternative argument from cor.test of stats package. Can be "two.sided", "less" or "greater".

Value

Returns a matrix with the correlation values and the p-values

Examples

```
## Not run:
  ## Example of correlations test of variables (computationally heavy)
  data(cachexia)
  corr.result = correlations_test(cachexia,
method = "pearson", by.var = FALSE)

## End(Not run)
```

correlation_test *Correlation test of two variables or samples*

Description

Performs correlations test of two variables or samples from the dataset.

Usage

```
correlation_test(dataset, x, y, method = "pearson",
alternative = "two.sided", by.var = T)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
x	first variable or sample.
y	second variable or sample.
method	correlation method, it can be "pearson", "kendall" or "spearman".
alternative	alternative argument from cor.test of stats package. Can be "two.sided", "less" or "greater".
by.var	if TRUE then the correlations of the variables will be calculated, if not then the correlations of the samples will be calculated.

Value

Returns the correlation result from cor.test function of stats package.

Examples

```
  ## Example of correlations test of variables
  data(cachexia)
  corr.result = correlation_test(cachexia, "Serine", "Creatine", method = "pearson",
by.var = TRUE)
```

---

count_missing_values *Count missing values*

Description

Counts the missing values on the dataset.

Usage

```
count_missing_values(dataset)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.

Value

Returns the number of missing values on the dataset.

Examples

```
## Example of counting the missing values
data(cachexia)
count_missing_values(cachexia)
```

count_missing_values_per_sample

Count missing values per sample

Description

Counts the missing values on each sample of the dataset.

Usage

```
count_missing_values_per_sample(dataset, remove.zero = T)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
remove.zero	boolean value indicating if the results of samples with no missing value are removed.

Value

Returns a vector with the number of missing values on each sample.

Examples

```
## Example of counting the missing values on each sample
data(cachexia)
cachexia$data[10,10] = NA
count_missing_values_per_sample(cachexia)
```

count_missing_values_per_variable

Count missing values per variable

Description

Counts the missing values on each variable of the dataset.

Usage

```
count_missing_values_per_variable(dataset, remove.zero = T)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
remove.zero	boolean value indicating if the results of variables with no missing value are removed.

Value

Returns a vector with the number of missing values on each sample.

Examples

```
## Example of counting the missing values on each variable
data(cachexia)
cachexia$data[10,10] = NA
count_missing_values_per_variable(cachexia)
```

`create_dataset` *Create dataset*

Description

Create a dataset from existing objects

Usage

```
create_dataset(datamatrix, type = "undefined", metadata = NULL,
description = "", sample.names = NULL, x.axis.values = NULL,
label.x = NULL, label.values = NULL, xSet = NULL)
```

Arguments

`datamatrix`	matrix with numerical data: rows are assumed to be variables and columns assumed to be samples.
`type`	type of data: string that can be one of the following: • nmr-spectra • nmr-peaks • ir-spectra • uvv-spectra • raman-spectra • fluor-spectra • ms-spectra • lcms-peaks • gcms-peaks • integrated-data • concentrations • undefined
`metadata`	data frame with the dataset's metadata: columns represent each metadata variable and rows represent the value of the metadata for the sample.
`description`	string with a short description of the dataset.

sample.names	vector with sample names, if NULL then the column names of datamatrix or sequential numbers will be used.
x.axis.values	vector with the x axis values, if NULL then the row names of datamatrix or sequential numbers will be used.
label.x	x axis label.
label.values	values label.
xSet	xcmsSet object from xcms package to store the reading and preprocessing results from MS spectra. Used for metabolite identification purposes.

Value

list representing the dataset:

data	matrix with the data
type	type of the data
description	short description of the dataset
metadata	data frame with the metadata variables
labels	list with labels of x axis and values
xSet	xcmsSet object

Examples

```
## Not run:
  ## Example of creating a dataset
  dataset = create_dataset(datamatrix, type = "nmr-spectra",
metadata = dataset.metadata, description = "shortdescription",
sample.names = NULL, x.axis.values = NULL, label.x = "ppm",
label.values = "intensity")

## End(Not run)
```

---

cubic_root_transform *Cubic root transformation*

Description

Performs cubic root transformation on the data matrix.

Usage

```
cubic_root_transform(datamat)
```

Arguments

datamat	data matrix.

Value

Returns the data matrix with the cubic root transformation applied.

Examples

```
## Example of cubic root transformation
data(cachexia)
datamat.cubic = cubic_root_transform(cachexia$data)
```

`dataset_from_peaks` *Dataset from peaks*

Description

Converts a peak list to a dataset.

Usage

```
dataset_from_peaks(sample.list, metadata = NULL,
description = "", type = "nmr-peaks")
```

Arguments

`sample.list`	list with the peaks from each sample.
`metadata`	data frame with the associated metadata.
`description`	string with the description of the dataset.
`type`	string that represents the type of the data.

Value

Returns the dataset from the peak list.

Examples

```
## Not run:
    ## Example of converting a peak list to a dataset (computationally heavy)
    data(propolisSampleList)
    dataset = dataset_from_peaks(propolisSampleList, metadata = NULL,
      description = "some text", type = "nmr-peaks")

## End(Not run)
```

data_correction *Data correction*

Description

Perform spectra corrections with 3 different methods.

Usage

```
data_correction(dataset, type = "background",
method = "modpolyfit", ...)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
type	string that represents the type of correction that will be applied to the spectra. The three possible types are: "background", to perform background correction; "offset", to perform offset correction; and "baseline", to perform baseline correction.
method	string parameter of baseline correction indicating the correction method.
...	additional parameters of baseline correction.

Value

Returns the dataset with the spectra corrected.

Examples

```
## Example of data correction using background correction
data(cassavaPPD)
dataset.corrected = data_correction(cassavaPPD, type = "background")
```

dendrogram_plot *Plot dendrogram*

Description

Plot dendrogram of hierarchical clustering results.

Usage

```
dendrogram_plot(dataset, hc.result, column.metadata = 1,
labels = NULL, ...)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
hc.result	object of class hclust with the clustering results.
column.metadata	string or index indicating what metadata to use to name the leafs.
labels	vector with the leaf names (optional).
...	other parameters for plotting.

Examples

```
## Example of a dendrogram
data(cachexia)
hc.result = hierarchical_clustering(cachexia)
dendrogram_plot(cachexia, hc.result)
```

---

dendrogram_plot_col *Plot dendrogram*

Description

Plot dendrogram of hierarchical clustering results with different colors

Usage

```
dendrogram_plot_col(dataset, hc.result, classes.col, title = "",
lab.cex = 1, leg.pos = "topright",...)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
hc.result	object of class hclust with the clustering results.
classes.col	string or index indicating what metadata to use to name the leafs.
title	title of dendrogram.
lab.cex	the magnification to be used for x and y labels relative to the current setting of cex.
leg.pos	position of the legend.
...	other parameters for plotting.

Examples

```
## Example of colored dendrogram
data(cachexia)
hc.result = hierarchical_clustering(cachexia)
dendrogram_plot_col(cachexia, hc.result, "Muscle.loss",
title = "Example")
```

`feature_selection` *Perform feature selection*

Description

Perform feature selection on the dataset.

Usage

```
feature_selection(dataset, column.class, method = "rfe",
functions, validation = "cv", repeats = 5, number = 10,
subsets = 2^(2:4))
```

Arguments

`dataset`	list representing the dataset from a metabolomics experiment.
`column.class`	string or index indicating what metadata to use.
`method`	method used for feature selection. Possible values are "rfe" (recursive feature elimination) and "filter" (Selection by filter - sbf) from caret's package.
`functions`	a list of functions for model fitting, prediction and variable importance/filtering.
`validation`	the external resampling method: boot, cv, LOOCV or LGOCV (for repeated training/test splits.
`repeats`	for repeated k-fold cross-validation only: the number of complete sets of folds to compute.
`number`	either the number of folds or number of resampling iterations.
`subsets`	a numeric vector of integers corresponding to the number of features that should be retained (rfe only).

Value

caret's result from rfe or sbf.

Examples

```
## Not run:
  ## Example of feature selection using rfe and sbf
  data(cachexia)
  library(caret)
  rfe.result = feature_selection(cachexia, "Muscle.loss",
               method="rfe", functions = caret::rfFuncs,
               validation = "cv", number = 3,
               subsets = 2^(1:6))
  sbf.result = feature_selection(cachexia, "Muscle.loss",
       method="filter", functions = caret::rfSBF,
       validation = "cv")

## End(Not run)
```

`filter_feature_selection` *Perform selection by filter*

Description

Perform selection by filter using univariate filters, from caret's package.

Usage

```
filter_feature_selection(datamat, samples.class,
functions = caret::rfSBF, method = "cv", repeats = 5)
```

Arguments

`datamat`	data matrix from dataset.
`samples.class`	string or index indicating what metadata to use.
`functions`	a list of functions for model fitting, prediction and variable filtering.
`method`	the external resampling method: boot, cv, LOOCV or LGOCV (for repeated training/test splits.
`repeats`	for repeated k-fold cross-validation only: the number of complete sets of folds to compute.

Value

A caret's sbf object with the result of selection by filter.

Examples

```
## Not run:
  ## Example of selection by filter
  data(cachexia)
  library(caret)
  rfe.result = filter_feature_selection(cachexia$data,
       cachexia$metadata$Muscle.loss, functions = caret::rfSBF,
       method = "cv")

## End(Not run)
```

find_equal_samples *Find equal samples*

Description

Finds samples that have the same peak values - x and y (equal data frames)

Usage

```
find_equal_samples(sample.list)
```

Arguments

sample.list	list of data frames with the samples' peaks.

Value

Returns a dataframe with two columns indicating which pair of samples are equal.

Examples

```
## Example of finding equal samples
data(propolisSampleList)
equal.samples = find_equal_samples(propolisSampleList)
```

first_derivative *First derivative*

Description

Calculates the first derivative of the data.

Usage

```
first_derivative(dataset)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.

Value

Return the dataset with the first derivative of the data calculated.

Examples

```
## Example of calculate the first derivative
data(cassavaPPD)
cassava.derivative = first_derivative(cassavaPPD)
```

`flat_pattern_filter` *Flat pattern filter*

Description

Performs a flat pattern filter over the dataset.

Usage

```
flat_pattern_filter(dataset, filter.function = "iqr",
by.percent = T, by.threshold = F, red.value = 0)
```

Arguments

`dataset`	list representing the dataset from a metabolomics experiment.
`filter.function`	filter function. It can be: • **"iqr"** - Interquantile Range • **"rsd"** - Relative Standard Deviation • **"sd"** - Standard Deviation • **"mad"** - Median Absolute Deviation • **"mean"** - Mean • **"median"** - Median
`by.percent`	boolean value, if T the number of variables to filter will be a percentage of the number of variables in the dataset; percentage is given by the "red.value" parameter
`by.threshold`	boolean value, if T, defines filtering will select variables where values of filtering function are below a given threshold. Threshold is defined by red.value that defines the minimum value of the function needed to keep the variable.
`red.value`	it can be the percentage or the threshold number. If red.value = "auto", will calculate number of variables to remove automatically

Value

Returns the dataset with the data filtered.

Examples

```
## Example of flat pattern filter
data(cassavaPPD)
dataset.filtered = flat_pattern_filter(cassavaPPD, "iqr", by.percent = TRUE,
   red.value = 20)
```

fold_change *Fold change analysis*

Description

Perform fold change analysis on the dataset.

Usage

```
fold_change(dataset, metadata.var, ref.value,
threshold.min.fc = NULL, write.file = F,
file.out = "fold_change.csv")
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
metadata.var	metadata to use to calculate the fold change.
ref.value	class name to indicate the initial value.
threshold.min.fc	minimum threshold of the fold change value.
write.file	boolean value to write or not a file with the results.
file.out	name of the file.

Value

Table of results with fold change and log2 of fold change.

Examples

```
  ## Example of fold change
  data(cachexia)
  fold.change.results = fold_change(cachexia, "Muscle.loss",
"control", write.file = FALSE)
```

fold_change_var *Fold change applied on two variables*

Description

Fold change applied on two variables. Instead of having the difference of the variables on two groups, we have the difference of the groups on two variables.

Usage

```
fold_change_var(dataset, metadata.var, variables,
threshold.min.fc = NULL, write.file = F,
file.out = "fold_change_reverse.csv")
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
metadata.var	metadata to use to calculate the fold change.
variables	vector with two positions containing the name of the variables.
threshold.min.fc	minimum threshold of the fold change value.
write.file	boolean value to write or not a file with the results.
file.out	name of the file.

Value

Table of results with fold change and log2 of fold change.

Examples

```
  ## Example of fold change reverse
  data(cachexia)
  fold.change.results = fold_change_var(cachexia, "Muscle.loss",
c("Creatine", "Serine"))
```

---

get_data *Get data*

Description

Get the data matrix from dataset

Usage

```
get_data(dataset)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.

Value

Returns the data matrix

Examples

```
## Example of getting the data matrix
data(cachexia)
cachexia.dm = get_data(cachexia)
```

`get_data_as_df` *Get data as data frame*

Description

Get the data matrix from the dataset as a data frame.

Usage

```
get_data_as_df(dataset)
```

Arguments

`dataset`	list representing the dataset from a metabolomics experiment.

Value

Returns the data matrix from the dataset as a data.frame object.

Examples

```
## Example of getting the data matrix as data frame
data(cachexia)
cachexia.dt = get_data_as_df(cachexia)
```

`get_data_value` *Get data value*

Description

Get a data value given the x-axis labels and the sample

Usage

```
get_data_value(dataset, x.axis.val, sample, by.index = F)
```

Arguments

`dataset`	list representing the dataset from a metabolomics experiment.
`x.axis.val`	index or name of the x-axis value.
`sample`	index or name of the sample.
`by.index`	boolean value indicating if the x-axis value and sample are represented as index or not.

Value

Returns a numeric with the data point value.

Examples

```
## Example of getting a data value from the dataset
data(cachexia)
data.value = get_data_value(cachexia, "Creatine", "PIF_178",
     by.index = FALSE)
```

`get_data_values` *Get data values*

Description

Gets the values of all samples in the dataset given a set of x axis names or indexes.

Usage

```
get_data_values(dataset, x.axis.val, by.index = FALSE)
```

Arguments

`dataset`	list representing the dataset from a metabolomics experiment.
`x.axis.val`	vector with the values of the x axis (could be names or indexes).
`by.index`	boolean value indicating if the x.axis.val is a vector of indexes or not.

Value

Returns a matrix with the values of all samples in the specified x axis.

Examples

```
## Example of getting a metadata value
data(cachexia)
data.values = get_data_values(cachexia, c("Creatine","Serine","Lactate"),
by.index = FALSE)
```

get_metadata *Get metadata*

Description

Get the metadata from the dataset

Usage

```
get_metadata(dataset)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.

Value

returns a data frame with the metadata.

Examples

```
## Example of getting the metadata
data(cachexia)
cachexia.mt = get_metadata(cachexia)
```

get_metadata_value *Get metadata value*

Description

Get the metadata value

Usage

```
get_metadata_value(dataset, variable, sample)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
variable	index or name of the metadata variable.
sample	index or name of the sample.

Value

Return the corresponding metadata value of the sample.

Examples

```
## Example of getting a metadata value
data(cachexia)
metadata.value = get_metadata_value(cachexia, "Muscle.loss", "PIF_178")
```

---

get_metadata_var *Get metadata variable*

Description

Get the values of a metadata variable from the dataset.

Usage

```
get_metadata_var(dataset, var)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
var	index or name of the metadata variable.

Value

Returns a vector with the values of the metadata variable.

Examples

```
## Example of getting a metadata variable
data(cachexia)
metadata.variable = get_metadata_var(cachexia, "Muscle.loss")
```

---

get_peak_values *Get peak values*

Description

Gets the peak values from a data frame of samples' peaks.

Usage

```
get_peak_values(samples.df, peak.val)
```

Arguments

samples.df	data frame with the samples' peaks.
peak.val	peak name.

Value

Returns a vector with the peak values.

Examples

```
## Example of getting the peak values
data(propolis)
peak.values = get_peak_values(propolis$data, 2.11)
```

`get_samples_names_dx` *Get sample's names from DX files*

Description

Function to get the names of the DX files from a folder.

Usage

```
get_samples_names_dx(foldername)
```

Arguments

`foldername`	string with the path of the data folder.

Value

Returns a vector with the sample's names.

Examples

```
## Not run:
  ## Example of getting DX sample's names
  dx.sample.names = get_samples_names_dx("data")

## End(Not run)
```

get_samples_names_spc *Get sample's names from SPC files*

Description

Function to get the names of the SPC files from a folder.

Usage

```
get_samples_names_spc(foldername)
```

Arguments

foldername	string with the path of the data folder.

Value

Returns a vector with the sample's names.

Examples

```
## Not run:
  ## Example of getting SPC sample's names
  spc.sample.names = get_samples_names_spc("data")

## End(Not run)
```

get_sample_names *Get sample names*

Description

Get the sample names from the dataset.

Usage

```
get_sample_names(dataset)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.

Value

Returns a vector with the sample names.

Examples

```
## Example of getting the sample names
data(cachexia)
sample.names = get_sample_names(cachexia)
```

`get_type` *Get type of data*

Description

Get the type of the data from the dataset

Usage

```
get_type(dataset)
```

Arguments

`dataset`	list representing the dataset from a metabolomics experiment.

Value

Returns a string with the type of the data.

Examples

```
## Example of getting the type of the data
data(cachexia)
type = get_type(cachexia)
```

`get_value_label` *Get value label*

Description

Get the value label from the dataset

Usage

```
get_value_label(dataset)
```

Arguments

`dataset`	list representing the dataset from a metabolomics experiment.

Value

Returns a string with the value label.

Examples

```
## Example of getting the value label
data(cassavaPPD)
value.label = get_value_label(cassavaPPD)
```

`get_x_label` *Get x-axis label*

Description

Get the x-axis label from the dataset.

Usage

```
get_x_label(dataset)
```

Arguments

`dataset`	list representing the dataset from a metabolomics experiment.

Value

Returns a string with the x-axis label.

Examples

```
## Example of getting the x-axis label
data(cassavaPPD)
x.label = get_x_label(cassavaPPD)
```

`get_x_values_as_num` *Get x-axis values as numbers*

Description

Get the x-axis values from the dataset as numbers.

Usage

```
get_x_values_as_num(dataset)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.

Value

Returns a numeric vector with the x-axis values, if the variable labels are not all numeric then an error message is shown.

Examples

```
## Example of getting the x-axis values as numbers
data(cassavaPPD)
xvalues.numeric = get_x_values_as_num(cassavaPPD)
```

---

get_x_values_as_text *Get x-axis values as text*

Description

Get the x-axis values from the dataset as text.

Usage

```
get_x_values_as_text(dataset)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.

Value

Returns a character vector with the x-axis values.

Examples

```
## Example of getting the x-axis values as text
data(cassavaPPD)
xvalues.text = get_x_values_as_text(cassavaPPD)
```

group_peaks *Group peaks*

Description

Group peaks with peak alignment.

Usage

```
group_peaks(sample.list, type, method = "own", metadata = NULL,
samp.classes = 1, description = "", label.x = NULL,
label.values = NULL, step = 0.03)
```

Arguments

sample.list	list containing the sample's data.
type	type of the data.
method	method of peak alignment. Can be "own" or "metaboanalyst", which the later is for using the peak alignment used in MetaboAnalyst software.
metadata	data frame containing the metadata.
samp.classes	the metadata's variable to be used in the MetaboAnalyst method.
description	short description of the data.
label.x	the label for the x values.
label.values	the label for the y values.
step	step value for the peak alignment process.

Value

Returns a dataset with the peaks of the data aligned.

Examples

```
## Not run:
   ## Example of grouping peaks (computationally heavy)
   data(propolisSampleList)
   peaks.ds = group_peaks(propolisSampleList, "nmr-peaks", method = "own",
    metadata = NULL, description = "short description",
label.x = "ppm", label.values = "intensity", step = 0.03)

## End(Not run)
```

heatmap_correlations *Correlations heatmap*

Description

Plots a heatmap with the correlations.

Usage

```
heatmap_correlations(correlations, col = NULL, ...)
```

Arguments

correlations	correlation matrix
col	colors to be used on heatmap.
...	extra parameters to visual purposes.

Examples

```
## Example of correlations heatmap
data(cachexia)
correlations = correlations_dataset(cachexia)
heatmap_correlations(correlations)
```

hierarchical_clustering *Perform hierarchical clustering analysis*

Description

Perform hierarchical clustering analysis on the dataset.

Usage

```
hierarchical_clustering(dataset, distance = "euclidean",
clustMethod = "complete", hc.type = "samples")
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
distance	the distance measure to be used to compute the distances between the rows of a data matrix. Possible types are "euclidean", "manhattan", "pearson" or "spearman".
clustMethod	the agglomeration method to be used. Possible values are "ward", "single", "complete", "average", "mcquitty", "median" or "centroid".
hc.type	a string indicating if hierarchical cluster analysis will be performed on samples ("samples") or on variables ("variables")

Value

An object of class hclust with the clustering results.

Examples

```
## Example of hierarchical clustering
data(cachexia)
hc.result = hierarchical_clustering(cachexia,
    distance = "euclidean", clustMethod = "complete",
    hc.type = "samples")
```

`impute_nas_knn` *Impute missing values with KNN*

Description

Impute missing values with KNN

Usage

```
impute_nas_knn(dataset, k = 10, ...)
```

Arguments

`dataset`	list representing the dataset from a metabolomics experiment.
`k`	the number of nearest neighbors.
`...`	additional values to impute.knn function.

Value

Returns the dataset with no missing values.

Examples

```
## Example of NA imputation with knn
data(propolis)
dataset = impute_nas_knn(propolis, k=10)
```

impute_nas_linapprox *Impute missing values with linear approximation*

Description

Impute missing values with linear approximation.

Usage

```
impute_nas_linapprox(dataset)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.

Value

Returns the dataset with no missing values.

Examples

```
## Example of NA imputation with linear approximation
data(propolis)
dataset = impute_nas_linapprox(propolis)
```

impute_nas_mean *Impute missing values with mean*

Description

Impute missing values with mean

Usage

```
impute_nas_mean(dataset)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.

Value

Returns the dataset with no missing values.

Examples

```
## Example of NA imputation with mean
data(propolis)
propolis = impute_nas_mean(propolis)
```

impute_nas_median *Impute missing values with median*

Description

Impute missing values with median

Usage

```
impute_nas_median(dataset)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.

Value

Returns the dataset with no missing values.

Examples

```
## Example of NA imputation with median
data(propolis)
propolis = impute_nas_median(propolis)
```

impute_nas_value *Impute missing values with value replacement*

Description

Impute missing values with value replacement.

Usage

```
impute_nas_value(dataset, value)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
value	value to replace the missing values.

Value

Returns the dataset with no missing values.

Examples

```
## Example of NA imputation with value replacing
data(propolis)
propolis = impute_nas_value(propolis, 0.0005)
```

`indexes_to_xvalue_interval` *Get the x-values of a vector of indexes*

Description

Returns x-values corresponding to a vector of indexes (only to numerical values - spectra)

Usage

```
indexes_to_xvalue_interval(dataset, indexes)
```

Arguments

`dataset`	list representing the dataset from a metabolomics experiment.
`indexes`	numeric vector containing the indexes.

Value

Returns a numeric vector with the interval of x-values from the indexes vector

Examples

```
## Example of getting the interval of x-values from indexes
data(cassavaPPD)
xvalue.interval = indexes_to_xvalue_interval(cassavaPPD, c(10,50))
```

`is_spectra` *Check type of data*

Description

Check if the dataset is from spectral data where x.values are numeric.

Usage

```
is_spectra(dataset)
```

Arguments

`dataset`	list representing the dataset from a metabolomics experiment.

Value

Returns a boolean indicating if the dataset is from spectral data or not.

Examples

```
## Example of checking if the dataset is from spectral data
data(cassavaPPD)
is_spectra(cassavaPPD)
```

`kmeans_clustering` *Perform k-means clustering analysis*

Description

Perform k-means clustering analysis on the dataset.

Usage

```
kmeans_clustering(dataset, num.clusters, type = "samples")
```

Arguments

`dataset`	list representing the dataset from a metabolomics experiment.
`num.clusters`	the number of clusters.
`type`	a string indicating if k-means will be performed on samples ("samples") or on variables ("variables")

Value

An object of class kmeans with the clustering results.

Examples

```
## Example of kmeans clustering
data(cachexia)
kmeans.result = kmeans_clustering(cachexia,
num.clusters = 4, type = "samples")
```

kmeans_plot *Plot kmeans clusters*

Description

Plot for each formed cluster, in grey the values of all samples of that cluster and in blue the median of that samples.

Usage

```
kmeans_plot(dataset, kmeans.result)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
kmeans.result	object of class kmeans with the clustering results.

Examples

```
  ## Example of kmeans plot - dataset filtered for performance purposes
  data(cassavaPPD)
  cassavaPPD = flat_pattern_filter(cassavaPPD, "iqr", by.percent = TRUE,
red.value = 70)
  kmeans.result = kmeans_clustering(cassavaPPD, 3)
  kmeans_plot(cassavaPPD, kmeans.result)
```

kmeans_result_df *Show cluster's members*

Description

Show for each cluster from kmeans analysis the sample names belonging to them.

Usage

```
kmeans_result_df(kmeans.result)
```

Arguments

kmeans.result	object of class kmeans with the clustering results.

Value

Data frame with the clusters and the samples' names that belong to each one.

Examples

```
## Example of showing kmeans cluster's members
data(cassavaPPD)
kmeans.result = kmeans_clustering(cassavaPPD, 3)
kmeans_result_df(kmeans.result)
```

---

kruskalTest_dataset *Kruskal-Wallis tests on dataset*

Description

Run Kruskal-Wallis Tests for each row of the data from the dataset.

Usage

```
kruskalTest_dataset(dataset, metadata.var, threshold = NULL,
write.file = F, file.out = "kruskal.csv")
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
metadata.var	metadata variable to use in the t-tests.
threshold	threshold value of the p-value.
write.file	boolean value to write or not a file with the results.
file.out	name of the file.

Value

Table with the results of the Kruskal-Wallis tests, with p-value, -log10(p-value) and false discovery rate (fdr).

Examples

```
## Example of ks-Tests on dataset
data(cachexia)
kruskaltests.result = kruskalTest_dataset(cachexia, "Muscle.loss",
write.file = FALSE)
```

ksTest_dataset *Kolmogorov-Smirnov tests on dataset*

Description

Run Kolmogorov-Smirnov Tests for each row of the data from the dataset.

Usage

```
ksTest_dataset(dataset, metadata.var, threshold = NULL,
write.file = F, file.out = "ks.csv")
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
metadata.var	metadata variable to use in the t-tests.
threshold	threshold value of the p-value.
write.file	boolean value to write or not a file with the results.
file.out	name of the file.

Value

Table with the results of the Kolmogorov-Smirnov tests, with p-value, -log10(p-value) and false discovery rate (fdr).

Examples

```
## Example of ks-Tests on dataset
data(cachexia)
kstests.result = ksTest_dataset(cachexia, "Muscle.loss",
write.file = FALSE)
```

linregression_onevar *Linear regression on one variable*

Description

Performs linear regression on one variable of the dataset.

Usage

```
linregression_onevar(dataset, x.val, metadata.vars, combination)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
x.val	the x-value to be tested.
metadata.vars	metadata variables to use in linear regression.
combination	a formula specifying the model.

Value

Returns a summary of the result from the lm function from stats package.

Examples

```
 ## Not run:
  ## Example of linear regression on all variables
  linreg.results = linregression_onevar(metabolomics.dataset, "200",
   c("metadata1","metadata2"), "metadata1+metadata2")

## End(Not run)
```

---

linreg_all_vars *Linear Regression*

Description

Performs linear regression analysis over the dataset with the selected metadata's variables.

Usage

```
linreg_all_vars(dataset, metadata.vars, combination)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
metadata.vars	metadata variables to use in linear regression.
combination	a formula specifying the model.

Value

Returns a list where each element is the linear regression result of a variable on the dataset.

Examples

```
## Not run:
  ## Example of linear regression on all variables
  linreg.results = linreg_all_vars(metabolomics.dataset, c("metadata1",
   "metadata2"), "metadata1+metadata2")

## End(Not run)
```

linreg_coef_table *Linear regression coefficient table*

Description

Gets a data.frame with the coefficient values.

Usage

```
linreg_coef_table(linreg.results, write.file = F,
file.out = "linreg-coefs.csv")
```

Arguments

linreg.results	Linear regression results from linreg.all.vars function.
write.file	boolean value to indicate if a file should be written with the results.
file.out	name of the file.

Value

Returns a data.frame with the coefficient values.

Examples

```
## Not run:
  ## Example of linear regression coefficient table
  linreg.coef.tab = linreg_coef_table(linreg.results)

## End(Not run)
```

linreg_pvalue_table *Linear regression p-values table*

Description

Gets the p-values table from the linear regression analysis.

Usage

```
linreg_pvalue_table(linreg.results, write.file = F,
file.out = "linreg-pvalues.csv")
```

Arguments

linreg.results	Linear regression results from linreg.all.vars function.
write.file	boolean value to indicate if a file should be written with the results.
file.out	name of the file.

Value

Returns a data.frame with the p-values.

Examples

```
## Not run:
  ## Example of linear regression p-values table
  linreg.pvalue.tab = linreg_pvalue_table(linreg.results)

## End(Not run)
```

`linreg_rsquared` *Linear regression r-squared*

Description

Gets the linear regression r-squared values.

Usage

```
linreg_rsquared(linreg.results, write.file = F,
file.out = "linreg-rsquared.csv")
```

Arguments

`linreg.results`	Linear regression results from linreg.all.vars function.
`write.file`	boolean value to indicate if a file should be written with the results.
`file.out`	name of the file.

Value

Returns a data.frame with the r-squared values.

Examples

```
## Not run:
  ## Example of linear regression rsquared values and the adjusted
  ## rsquared values
  linreg.rsquared.tab = linreg_rsquared(linreg.results)

## End(Not run)
```

log_transform *Logarithmic transformation.*

Description

Performs logarithmic transformation on the data matrix.

Usage

```
log_transform(datamat)
```

Arguments

datamat	data matrix.

Value

Returns the data matrix with the logarithmic transformation applied.

Examples

```
## Example of logarithmic transformation
data(cassavaPPD)
datamat.log = log_transform(cassavaPPD$data)
```

low_level_fusion *Low level fusion*

Description

Low level fusion method for integrate different datasets (only samples with the same name on all datasets will be merged)

Usage

```
low_level_fusion(datasets)
```

Arguments

datasets	list containing the datasets to be fused (each dataset is a list from the pre defined format for the dataset).

Value

Return a single dataset with all the datasets merged.

Examples

```
## Not run:
  ## Example of low level fusion
  datasets = list(dataset1, dataset2)
  dataset.fusion = low_level_fusion(datasets)

## End(Not run)
```

`MAIT_identify_metabolites` *MAIT metabolite identification*

Description

Performs metabolite identification using MAIT.

Usage

```
MAIT_identify_metabolites(dataset, metadata.variable,
xSet = NULL, data.folder = NULL, features = NULL,
mass.tolerance = 0.5)
```

Arguments

`dataset`	list representing the dataset from a metabolomics experiment.
`metadata.variable`	metadata's variable.
`xSet`	xcmsSet object that can be passed.
`data.folder`	string indicating the data folder.
`features`	features that can be used to help to identify the metabolites.
`mass.tolerance`	mass tolerance.

Value

Returns an object resulted from identifyMetabolites function from MAIT package.

References

http://www.bioconductor.org/packages/release/bioc/html/MAIT.html

Examples

```
## Not run:
  ## Example of MAIT metabolite identification
  data(spinalCord)
  library(MAIT)
  mait.metabolites = MAIT_identify_metabolites(spinalCord, "type",
features = "all", data.folder = "data",
xSet = spinalCord$xSet)
  mait.metab.table = mait.metabolites@FeatureInfo@metaboliteTable
  mait.metab.table[which(mait.metab.table$Name != "Unknown"),
  c(1,3,6)]

## End(Not run)
```

`mean_centering` *Mean centering*

Description

Performs mean centering on the dataset.

Usage

```
mean_centering(dataset)
```

Arguments

`dataset`	list representing the dataset from a metabolomics experiment.

Value

Returns the dataset with mean centering applied.

Examples

```
  ## Example of mean centering
  data(cassavaPPD)
  cassavaPPD = mean_centering(cassavaPPD)
```

merge_datasets *Merge two datasets*

Description

Merges two datasets with the same variables and metadata's variables.

Usage

```
merge_datasets(dataset1, dataset2)
```

Arguments

dataset1	list representing the first dataset from a metabolomics experiment.
dataset2	list representing the second dataset from a metabolomics experiment.

Value

Returns one dataset with the data from the two datasets merged.

Examples

```
## Not run:
  ## Example of merging two datasets
  dataset = merge_datasets(dataset1, dataset2)

## End(Not run)
```

merge_data_metadata *Merge data and metadata*

Description

Merges the data and metadata from the dataset into a single data.frame.

Usage

```
merge_data_metadata(dataset, samples = NULL,
metadata.vars = NULL, x.values = NULL, by.index = F)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
samples	vector with indexes or names of the samples to select
metadata.vars	metadata's variables.
x.values	vector with the desired x-values to get from the dataset.
by.index	if TRUE, the values of the x.values argument are indexes.

Value

Returns a data.frame with the data and metadata from the dataset merged.

Examples

```
## Example of merging data and metadata
data(cachexia)
dt.merged = merge_data_metadata(cachexia)
```

`metadata_as_variables` *Metadata as variables*

Description

Use one or more metadata variables as variables.

Usage

```
metadata_as_variables(dataset, metadata.vars, by.index = F)
```

Arguments

`dataset`	list representing the dataset from a metabolomics experiment.
`metadata.vars`	name or index of the metadata variables that are going to be used as data variables.
`by.index`	boolean value indicating if the metadata variables are indexes or names

Value

Returns the dataset with the metadata variables removed from the metadata and added on the data matrix.

Examples

```
## Example of using a metadata variable as data variable
data(propolis)
propolis = metadata_as_variables(propolis, "seasons", by.index = FALSE)
```

missingvalues_imputation *Missing values imputation*

Description

Treats the missing values of a dataset according to a specific method.

Usage

```
missingvalues_imputation(dataset, method = "value",
value = 5e-04, k = 5)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
method	imputation method. It can be: • **"value"** - replaces the missing values with a specific value • **"mean"** - replaces the missing values with the mean of the variables' values • **"median"** - replaces the missing values with the median of the variables' values • **"knn"** - replaces the missing values with k nearest neighbor averaging • **"linapprox"** - replaces the missing values with linear approximation
value	the value to replace the missing values if the method is "value".
k	the number of neighbors if the method is "knn".

Value

Returns the dataset with no missing values.

Examples

```
## Example of impute missing values
data(propolis)
dataset = missingvalues_imputation(propolis, method = "value",
  value = 0.0005)
```

msc_correction *Multiplicative scatter correction*

Description

Perform multiplicative scatter correction on the spectra.

Usage

```
msc_correction(dataset)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.

Value

Return the dataset with the multiplicative scatter correction employed on the data.

Examples

```
## Example of multiplicative scatter correction
data(cassavaPPD)
propolis.msc = msc_correction(cassavaPPD)
```

multiClassSummary *Multi Class Summary*

Description

Summary function for caret to compute AUC.

Usage

```
multiClassSummary(data, lev = NULL, model = NULL)
```

Arguments

data	data parameter.
lev	lev parameter.
model	model parameter.

References

www.r-bloggers.com/error-metrics-for-multi-class-problems-in-r-beyond-accuracy-and-kappa/

multifactor_aov_all_vars
Multifactor ANOVA

Description

Perform multi-factor ANOVA on all variables with the selected metadata variables.

Usage

```
multifactor_aov_all_vars(dataset, metadata.vars, combination)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
metadata.vars	metadata variables to use in ANOVA.
combination	a formula specifying the model.

Value

List where each element is the multifactor anova result of a variable on the dataset.

Examples

```
## Example of multifactor ANOVA on all variables
data(propolis)
propolis = missingvalues_imputation(propolis, "value", value = 0.00005)
m.aov.results = multifactor_aov_all_vars(propolis,
c("seasons","agroregions"), "seasons*agroregions")
```

multifactor_aov_pvalues_table
Multifactor ANOVA p-values table

Description

Gets the p-values table from the multifactor ANOVA results.

Usage

```
multifactor_aov_pvalues_table(multifactor.aov.results,
write.file = F, file.out = "multi-anova-pvalues.csv")
```

Arguments

multifactor.aov.results	multifactor anova results.
write.file	boolean value to indicate if a file is written.
file.out	name of the file.

Value

Returns a data.frame with the p-values.

Examples

```
## Example of multifactor ANOVA p-values table
data(propolis)
propolis = missingvalues_imputation(propolis, "value", value = 0.00005)
m.aov.results = multifactor_aov_all_vars(propolis,
c("seasons","agroregions"), "seasons*agroregions")
m.aov.pvalues = multifactor_aov_pvalues_table(m.aov.results)
```

---

multifactor_aov_varexp_table

Multifactor ANOVA variability explained table

Description

Gets the variability explained table from the multifactor ANOVA results.

Usage

```
multifactor_aov_varexp_table(multifactor.aov.results,
write.file = F, file.out = "multi-anova-varexp.csv")
```

Arguments

multifactor.aov.results	multifactor anova results.
write.file	boolean value to indicate if a file is written.
file.out	name of the file.

Value

Returns a data.frame with the variability explained.

Examples

```
## Example of multifactor ANOVA variability explained table
data(propolis)
propolis = missingvalues_imputation(propolis, "value", value = 0.00005)
m.aov.results = multifactor_aov_all_vars(propolis,
c("seasons","agroregions"), "seasons*agroregions")
m.aov.varepx = multifactor_aov_varexp_table(m.aov.results)
```

`multiplot` *Multiplot*

Description

Multiplot from ggplot2 - function taken from (see references).

Usage

```
multiplot(plots, plotlist = NULL, file, cols = 1, layout = NULL)
```

Arguments

`plots`	list with the plots to display.
`plotlist`	plot list.
`file`	file.
`cols`	number of columns.
`layout`	layout of the plot.

References

http://www.cookbook-r.com/Graphs/Multiple_graphs_on_one_page_

Examples

```
    ## Example of multiplot
    data(cachexia)
    pca.result = pca_analysis_dataset(cachexia)
    plot1 = pca_scoresplot2D(cachexia, pca.result, "Muscle.loss",
ellipses = TRUE)
    plot2 = pca_scoresplot2D(cachexia, pca.result, "Muscle.loss",
ellipses = FALSE, labels = TRUE)
    plts = list(plot1,plot2)
    multiplot(plts, cols = 2)
```

normalize *Normalize data*

Description

Normalize the data from the dataset with a specific method.

Usage

```
normalize(dataset, method, ref = NULL, constant = 1000)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
method	string specifying the normalization method. The possible values are: • **"sum"** normalization by sum. • **"median"** normalization by median. • **"ref.sample"** normalization by reference sample. • **"ref.feature"** normalization by reference feature.
ref	the reference if method is "ref.sample" or "ref.feature".
constant	the constant value if method is "sum".

Value

Returns the dataset with the data normalized.

Examples

```
## Example of normalization by median
data(cassavaPPD)
cassava.norm = normalize(cassavaPPD, "median")
```

normalize_samples *Normalize samples*

Description

Normalize the data from a datamatrix with a specific method.

Usage

```
normalize_samples(datamat, method, ref = NULL, constant = 1000)
```

Arguments

datamat	data matrix.
method	string specifying the normalization method. The possible values are: • **"sum"** normalization by sum. • **"median"** normalization by median. • **"ref.sample"** normalization by reference sample. • **"ref.feature"** normalization by reference feature.
ref	the reference if method is "ref.sample" or "ref.feature".
constant	the constant value if method is "sum".

Value

Returns the data matrix normalized.

Examples

```
## Example of normalization by median
data(cassavaPPD)
datamat.norm = normalize_samples(cassavaPPD$data, "median")
```

---

num_samples *Get number of samples*

Description

Get the number of samples from a dataset.

Usage

```
num_samples(dataset)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.

Value

Returns an integer with the number of samples in the dataset.

Examples

```
## Example of getting the number of samples
data(cachexia)
number.of.samples = num_samples(cachexia)
```

num_x_values *Get number of x values*

Description

Get the number of x-axis values.

Usage

```
num_x_values(dataset)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.

Value

Returns an integer with the number of x-axis values.

Examples

```
## Example of getting the number of x-axis values
data(cassavaPPD)
number.x.values = num_x_values(cassavaPPD)
```

offset_correction *Offset correction*

Description

Perform offset correction on the data.

Usage

```
offset_correction(dataset)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.

Value

Returns the dataset

Examples

```
## Example of offset correction
data(cassavaPPD)
cassava.corrected = offset_correction(cassavaPPD)
```

pca_analysis_dataset *PCA analysis (classical)*

Description

Performs a classical PCA analysis over the dataset.

Usage

```
pca_analysis_dataset(dataset, scale = T, center = T,
write.file = F, file.out = "pca", ...)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
scale	boolean value indicating if the variables are going to be scaled or not.
center	booleam value indicating if the variables are going to be centered or not.
write.file	boolean value that indicates if the results from PCA analysis are going to be written on a file.
file.out	name of the file that will store the results.
...	additional parameters to ggplot function.

Value

object of class 'prcomp' with the results from the PCA analysis.

Examples

```
## Example of performing a classical PCA analysis
data(cachexia)
pca.results = pca_analysis_dataset(cachexia)
```

pca_biplot *PCA biplot*

Description

Shows a PCA biplot.

Usage

```
pca_biplot(dataset, pca.result, cex = 0.8, legend.cex = 0.8,
x.colors = 1, inset = c(0, 0), legend.place = "topright", ...)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
pca.result	prcomp object with the PCA results.
cex	cex value.
legend.cex	cex value of the legend.
x.colors	colors of a metadata's variable.
inset	inset parameter of legend function.
legend.place	legend place.
...	additional parameters passed to biplot function.

Examples

```
## Example of a PCA biplot
data(cachexia)
pca.result = pca_analysis_dataset(cachexia)
pca_biplot(cachexia, pca.result, cex = 0.8)
```

---

pca_biplot3D *3D PCA biplot (interactive)*

Description

Shows a interactive 3D PCA biplot.

Usage

```
pca_biplot3D(dataset, pca.result, column.class = NULL,
pcas = c(1, 2, 3))
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
pca.result	prcomp object with the PCA results.
column.class	metadata's variable.
pcas	the three principal components.

Examples

```
## Example of a 3D PCA biplot
data(cachexia)
pca.result = pca_analysis_dataset(cachexia)
pca_biplot3D(cachexia, pca.result, "Muscle.loss", pcas = c(1,2,3))
```

pca_importance *PCA importance*

Description

Gets the importance from the PCs.

Usage

```
pca_importance(pca.res, pcs = 1:length(pca.res$sdev), sd = T,
prop = T, cumul = T, min.cum = NULL)
```

Arguments

pca.res	prcomp object with the PCA results.
pcs	vector with the PCs to get.
sd	boolean value indicating if standard deviation will be returned or not.
prop	boolean value that indicates if the proportion of variance is returned or not.
cumul	boolean value that indicates if the cumulative variance is returned or not.
min.cum	allows to define minimum cumulative % of variance

Value

Returns the information about the importance of the PCs.

Examples

```
## Example of performing a classical PCA analysis
data(cachexia)
pca.result = pca_analysis_dataset(cachexia)
pca_importance(pca.result, pcs = 1:5)
```

pca_kmeans_plot2D *2D PCA k-means plot*

Description

Groups the points with the clusters given by k-means in a 2D PCA scores plot.

Usage

```
pca_kmeans_plot2D(dataset, pca.result, num.clusters = 3,
pcas = c(1, 2), kmeans.result = NULL, labels = FALSE,
ellipses = FALSE, leg.pos = "right", xlim = NULL, ylim = NULL)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
pca.result	prcomp object with the PCA results.
num.clusters	number of clusters of k-means.
pcas	vector with the principal components to be plotted.
kmeans.result	result from k-means. If null k-means is performed in the function.
labels	boolean value indicating if the samples' labels will be shown.
ellipses	boolean value that indicates if an ellipse will be drawn on each group of the metadata's variable.
leg.pos	legend position.
xlim	vector with two positions with the x-axis limits.
ylim	vector with two positions with the y-axis limits.

Examples

```
## Not run:
  ## Example of a 2D PCA k-means plot
  data(cachexia)
  pca.result = pca_analysis_dataset(cachexia)
  pca_kmeans_plot2D(cachexia, pca.result, num.clusters = 3, pcas = c(1,2))

## End(Not run)
```

---

pca_kmeans_plot3D *3D PCA k-means plot (interactive)*

Description

Groups the points with the clusters given by k-means in a interactive 3D PCA scores plot.

Usage

```
pca_kmeans_plot3D(dataset, pca.result, num.clusters = 3,
pcas = c(1, 2, 3), kmeans.result = NULL, labels = FALSE,
size = 1, ...)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
pca.result	prcomp object with the PCA results.
num.clusters	number of clusters of k-means.
pcas	vector with the principal components to be plotted.
kmeans.result	result from k-means. If null k-means is performed in the function.
labels	boolean value indicating if the samples' labels will be shown.
size	parameter of plot3d from rgl package.
...	additional parameters of plot3d function from rgl package.

Examples

```
## Example of a 3D PCA k-means plot
data(cachexia)
pca.result = pca_analysis_dataset(cachexia)
pca_kmeans_plot3D(cachexia, pca.result, num.clusters = 3,
       pcas = c(1,2,3))
```

`pca_pairs_kmeans_plot` *PCA k-means pairs plot*

Description

Groups the points with the clusters from k-means in a PCA pairs plot.

Usage

```
pca_pairs_kmeans_plot(dataset, pca.result, num.clusters = 3,
kmeans.result = NULL, pcas = c(1, 2, 3, 4, 5))
```

Arguments

`dataset`	list representing the dataset from a metabolomics experiment.
`pca.result`	prcomp object with the PCA results.
`num.clusters`	number of clusters of k-means.
`kmeans.result`	result from k-means. If null k-means is performed in the function.
`pcas`	vector with the principal components to be plotted.

Examples

```
## Not run:
  ## Example of a PCA k-means pairs plot (computationally heavy)
  data(cachexia)
  pca.result = pca_analysis_dataset(cachexia)
  kmeans.res = kmeans_clustering(cachexia, 3)
  pca_pairs_kmeans_plot(cachexia, pca.result, num.clusters = 3,
   kmeans.result = kmeans.res, pcas = c(1,2,3,4,5))

## End(Not run)
```

pca_pairs_plot *PCA pairs plot*

Description

Shows a PCA pairs plot.

Usage

```
pca_pairs_plot(dataset, pca.result, column.class = NULL,
pcas = c(1, 2, 3, 4, 5), ...)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
pca.result	prcomp object with the PCA results.
column.class	metadata's variable.
pcas	the principal components to be shown.
...	additional parameters to ggpairs function from GGally package.

Examples

```
## Example of a PCA pairs plot
data(cachexia)
pca.result = pca_analysis_dataset(cachexia)
pca_pairs_plot(cachexia, pca.result, "Muscle.loss", pcas = c(1,2,3))
```

pca_plot_3d *3D pca plot*

Description

3D plot from 3 components

Usage

```
pca_plot_3d(dataset, model, var.class, pcas = 1:3, colors = NULL,
legend.place = "topright", ...)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
model	model with pca scores (pls model).
var.class	metadata column class.
pcas	the components to be plotted.
colors	colors of the groups.
legend.place	legend place.
...	additional parameters to legend function.

Examples

```
## Not run:
  ## Example of a 3d pca plot
  data(cachexia)
  train.result = train_models_performance(cachexia, "pls",
"Muscle.loss", "cv")
  pca_plot_3d(cachexia, train.result$final.models$pls, "Muscle.loss")

## End(Not run)
```

---

pca_robust *PCA analysis (robust)*

Description

Performs a robust PCA analysis.

Usage

```
pca_robust(dataset, center = "median", scale = "mad", k = 10,
write.file = F, file.out = "robpca", ...)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
center	indicates how the data is to be centered. Can be a function or a vector with the center values of each column.
scale	indicates how the data is to be rescaled. Can be a function or a vector with the scale value of each column.
k	the desired number of components to compute
write.file	boolean value that indicates if the results from PCA analysis are going to be written on a file.
file.out	name of the file that will store the results.
...	additional parameters pass to or from other functions.

Value

Returns an object of class 'princomp' with the PCA results.

Examples

```
  ## Example of performing a robust PCA analysis
  data(cachexia)
  pca.results = pca_robust(cachexia, center = "mean", scale = "mad",
k = 10)
```

`pca_scoresplot2D` *2D PCA scores plot*

Description

Shows a 2D PCA scores plot of two principal componets.

Usage

```
pca_scoresplot2D(dataset, pca.result, column.class,
pcas = c(1, 2), labels = FALSE, ellipses = FALSE,
pallette = 2, leg.pos = "right", xlim = NULL, ylim = NULL)
```

Arguments

`dataset`	list representing the dataset from a metabolomics experiment.
`pca.result`	prcomp object with the PCA results.
`column.class`	metadata's variable.
`pcas`	vector of two elements with the PCs that will be plotted.
`labels`	boolean value indicating if the sample's labels will be displayed.
`ellipses`	boolean value that indicates if an ellipse will be drawn on each group of the metadata's variable.
`pallette`	parameter of scale_colour_brewer from ggplot2.
`leg.pos`	position of the legend.
`xlim`	vector with two numeric values indicating the limits of the x axis.
`ylim`	vector with two numeric values indicating the limits of the y axis.

Examples

```
  ## Example of a 2D PCA scores plot
  data(cachexia)
  pca.result = pca_analysis_dataset(cachexia)
  pca_scoresplot2D(cachexia, pca.result, "Muscle.loss", pcas = c(1,2),
        ellipses = TRUE)
```

pca_scoresplot3D *3D PCA scores plot*

Description

Shows a 3D PCA scores plot of three principal componets.

Usage

```
pca_scoresplot3D(dataset, pca.result, column.class,
pcas = c(1, 2, 3))
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
pca.result	prcomp object with the PCA results.
column.class	metadata's variable.
pcas	vector with the principal components to be plotted.

Examples

```
## Example of a 3D PCA scores plot
data(cachexia)
pca.result = pca_analysis_dataset(cachexia)
pca_scoresplot3D(cachexia, pca.result, "Muscle.loss", pcas = c(1,2,3))
```

pca_scoresplot3D_rgl *3D PCA scores plot (interactive)*

Description

Shows a interactive 3D PCA scores plot of three principal components.

Usage

```
pca_scoresplot3D_rgl(dataset, pca.result, column.class,
pcas = c(1, 2, 3), size = 1, labels = FALSE)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
pca.result	prcomp object with the PCA results.
column.class	metadata's variable.
pcas	vector with the principal components to be plotted.
size	parameter of plot3d from rgl package.
labels	boolean value indicating if the samples' labels will be shown.

Examples

```
## Example of a 3D PCA scores plot
data(cachexia)
pca.result = pca_analysis_dataset(cachexia)
pca_scoresplot3D_rgl(cachexia, pca.result, "Muscle.loss",
        pcas = c(1,2,3), labels = TRUE)
```

`pca_screeplot` *PCA scree plot*

Description

PCA scree plot with the proportion and cumulative variance of the PCs.

Usage

```
pca_screeplot(pca.result, num.pcs = NULL, cex.leg = 0.8,
leg.pos = "right", lab.text = c("individual percent",
"cumulative percent"), fill.col = c("blue", "red"),
ylab = "Percentage", xlab = "Principal components", ...)
```

Arguments

`pca.result`	prcomp object with the PCA results.
`num.pcs`	number of principal components.
`cex.leg`	cex value of legend.
`leg.pos`	legend position.
`lab.text`	legend's labels.
`fill.col`	color of the legend's boxes.
`ylab`	y-axis label.
`xlab`	x-axis label
`...`	additional parameters to matplot.

Examples

```
## Example of a scree plot
data(cachexia)
pca.result = pca_analysis_dataset(cachexia)
pca_screeplot(pca.result)
```

peaks_per_sample *Peaks per sample*

Description

Counts number of peaks in a sample (given its index).

Usage

```
peaks_per_sample(sample.list, sample.index)
```

Arguments

sample.list	list of data frames with the samples' peaks.
sample.index	sample index.

Value

Return a integer value with the number of peaks in the sample.

Examples

```
## Example of counting the peaks in a sample
data(propolisSampleList)
num.peaks.sample = peaks_per_sample(propolisSampleList, 4)
```

peaks_per_samples *Peaks per samples*

Description

Calculates the number of peaks on each sample.

Usage

```
peaks_per_samples(sample.list)
```

Arguments

sample.list	list of data frames with the samples' peaks.

Value

Returns a numeric vector with the number of peaks on each sample.

Examples

```
## Example of counting the peaks in each sample
data(propolisSampleList)
num.peaks.samples = peaks_per_samples(propolisSampleList)
```

`plotvar_twofactor` *Plot variable distribution on two factors*

Description

Plot variable distribution on two factors from the dataset.

Usage

```
plotvar_twofactor(dataset, variable, meta.var1, meta.var2,
colour = "darkblue", title = "", xlabel = NULL, ylabel = NULL)
```

Arguments

`dataset`	list representing the dataset from a metabolomics experiment.
`variable`	variable's name.
`meta.var1`	first metadata's variable.
`meta.var2`	second metadata's variable.
`colour`	colours of the bodies of the boxplots.
`title`	title of the plot.
`xlabel`	x-axis label.
`ylabel`	y-axis label.

Value

Returns the plot object from ggplot function.

Examples

```
## Example of plotting a variable's distribution with 2 factors
data(cassavaPPD)
plotvar_twofactor(cassavaPPD, "399.3", "varieties", "ppds")
```

plot_anova *Plot ANOVA results*

Description

Function for plotting the results from ANOVA.

Usage

```
plot_anova(dataset, anova.results, anova.threshold = 0.01,
reverse.x = F)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
anova.results	ANOVA results.
anova.threshold	ANOVA threshold for the p-value.
reverse.x	boolean value to indicate if the x-axis is plotted in reverse.

Examples

```
    ## Example of plotting the ANOVA results - first filter the
    ## dataset to reduce computation time
    data(cassavaPPD)
    cassavaPPD = flat_pattern_filter(cassavaPPD, "iqr", by.percent = TRUE,
red.value = 75)
    anova.results = aov_all_vars(cassavaPPD, "varieties", doTukey = FALSE)
    plot_anova(cassavaPPD, anova.results, 0.05e-8)
```

plot_fold_change *Plot fold change results*

Description

Function for plotting the results from fold change.

Usage

```
plot_fold_change(dataset, fc.results, fc.threshold, plot.log = T,
var = F, xlab = "")
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
fc.results	fold change results.
fc.threshold	fold change threshold for the p-value.
plot.log	boolean value to determine if the fold change values are transformed logarithmically or not.
var	boolean value, if TRUE it uses the xlab argument to represent the xlabel of the plot.
xlab	string with the x axis description.

Examples

```
    ## Example of plotting the fold change results
    data(cachexia)
    fc.results = fold_change(cachexia, "Muscle.loss",
"control")
    plot_fold_change(cachexia, fc.results, 2)
```

---

plot_kruskaltest *Plot Kruskal-Wallis tests results*

Description

Function for plotting the results from Kruskal-Wallis tests.

Usage

```
plot_kruskaltest(dataset, kr.results, kr.threshold = 0.01)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
kr.results	Kruskal-Wallis tests results.
kr.threshold	Kruskal-Wallis test threshold for the p-value.

Examples

```
    ## Example of plotting the Kolmogorov-Smirnov tests results
    data(cachexia)
    kr.results = kruskalTest_dataset(cachexia, "Muscle.loss",
write.file = FALSE)
    plot_kruskaltest(cachexia, kr.results, 0.05)
```

plot_kstest *Plot Kolmogorov-Smirnov tests results*

Description

Function for plotting the results from Kolmogorov-Smirnov tests.

Usage

```
plot_kstest(dataset, ks.results, ks.threshold = 0.01)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
ks.results	Kolmogorov-Smirnov tests results.
ks.threshold	Kolmogorov-Smirnov test threshold for the p-value.

Examples

```
    ## Example of plotting the Kolmogorov-Smirnov tests results
    data(cachexia)
    ks.results = ksTest_dataset(cachexia, "Muscle.loss",
write.file = FALSE)
    plot_kstest(cachexia, ks.results, 0.05)
```

plot_regression_coefs_pvalues *Plot regression coefficient and p-values*

Description

Plots the linear regression coefficient and the p-values.

Usage

```
plot_regression_coefs_pvalues(linreg.results, bar.col = NULL,
coef.size = 5, ...)
```

Arguments

linreg.results	linear regression results.
bar.col	color of the bars.
coef.size	coefficient font size.
...	additional parameters to geom_text and geom_bar from ggplot.

Examples

```
    ## Not run:
      ## Example of multiplot
      plot_regression_coefs_pvalues(linreg.results)

## End(Not run)
```

`plot_spectra` *Plot spectra*

Description

Plot spectra from dataset.

Usage

```
plot_spectra(dataset, column.class, func = NULL, samples = NULL,
variable.bounds = NULL, xlab = NULL, ylab = NULL, lty = 1,
legend.place = "topright", cex = 0.8, reverse.x = F, ...)
```

Arguments

`dataset`	list representing the dataset from a metabolomics experiment.
`column.class`	string indicating the metadata's variable.
`func`	function to compute the summary statistics to apply to the data.
`samples`	vector with samples' names, if NULL all the samples will be considered.
`variable.bounds`	numeric vector with two elements indicating the interval of x-values to plot.
`xlab`	x-axis label.
`ylab`	y-axis label.
`lty`	parameter of matplot.
`legend.place`	string indicating the place that the legend's box will be placed.
`cex`	numeric value that indicates the amount by which the legend is magnified relative to the default.
`reverse.x`	boolean value indicating if the x-axis will be shown reversed or not.
`...`	additional parameters to matplot.

Examples

```
## Example of plotting spectra (simple)
data(cassavaPPD)
plot_spectra(cassavaPPD, "varieties", func = NULL,
 samples = c("BRA_1", "IAC5_4"), variable.bounds = c(1000,2000),
 legend.place = "topright")
```

plot_spectra_simple *Plot spectra (simple)*

Description

Plot spectra from dataset (simple version).

Usage

```
plot_spectra_simple(dataset, samples = NULL,
variable.bounds = NULL, xlab = NULL, ylab = NULL,
lty = 1, lwd = 1, col = 1, reverse.x = F, ...)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
samples	vector with samples' names, if NULL all the samples will be considered.
variable.bounds	numeric vector with two elements indicating the interval of x-values to plot.
xlab	x-axis label.
ylab	y-axis label
lty	parameter of matplot.
lwd	parameter of matplot.
col	parameter of matplot.
reverse.x	boolean value indicating if the x-axis will be shown reversed or not.
...	additional parameters to pass to matplot.

Examples

```
## Example of plotting spectra (simple)
data(cassavaPPD)
plot_spectra_simple(cassavaPPD, samples = c("IAC5_4", "BRA_1"),
variable.bounds = c(1000,2000))
```

plot_ttests *Plot t-tests results*

Description

Function for plotting the results from t-tests.

Usage

```
plot_ttests(dataset, tt.results, tt.threshold = 0.01)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
tt.results	t-tests results.
tt.threshold	t-test threshold for the p-value.

Examples

```
## Example of plotting the t-tests results
data(cachexia)
ttests.results = tTests_dataset(cachexia, "Muscle.loss")
plot_ttests(cachexia, ttests.results, 0.05)
```

predict_samples *Predict samples*

Description

Predict new samples.

Usage

```
predict_samples(train.result, new.samples)
```

Arguments

train.result	result from training a classifier.
new.samples	dataframe with new samples.

Value

Returns a data frame with the samples and the predicted class.

Examples

```
  ## Example of predicting samples
  data(cachexia)
  training.result = train_models_performance(cachexia, "pls",
"Muscle.loss", "cv")
  result = predict_samples(training.result$final.models$pls, cachexia$data)
```

`propolis` *Brazilian Propolis from different Harvest Seasons and different Agroecological Regions (dataset)*

Description

Propolis or bee glue is a sticky dark-colored substance produced from the collected buds or exudates of plants (resin) by bees (Apis mellifera L.). The resin is masticated, salivary enzymes are added, and the partially digested material is mixed with beewax and used in the hive to seal the walls, strengthen the borders of combs, and embalm dead invaders (Wollenweber et al., 1990). The propolis samples are from NMR data and were collected in the autumn (AU), winter (WI), spring (SP), and summer (SM) of 2010 from Apis mellifera hives located in southern Brazil (Santa Catarina State). A total of 59 samples were collected, and the distribution of samples by seasons being: SM - 16 samples, AU and SP - 15 samples, WI - 13 samples. Also, three agroecological regions were defined for the different apiaries, and one distributed as follows: Highlands - 12 samples, Plain - 11 samples, Plateau - 36 samples.

Usage

```
data(propolis)
```

Format

An object of class `"list"`

References

E. Wollenweber, B. M. Hausen, and W. Greenaway. Phenolic constituents and sensitizing properties of propolis, poplar balsam and balsam of peru. Bulletin de Groupe Polyphenol, 15:112-120, 1990. M. Maraschin, A. Somensi-Zeggio, S. K. Oliveira, S. Kuhnen, M. M. Tomazzoli, A. C. M. Zeri, R. Carreira, and M. Rocha. A machine learning and chemometrics assisted interpretation of spectroscopic data - a nmr-based metabolomics platform for the assessment of brazilian propolis. 2012

Examples

```
data(propolis)
sum_dataset(propolis)
```

`propolisSampleList` *Brazilian Propolis from different Harvest Seasons and different Agroecological Regions (sample list)*

Description

Propolis or bee glue is a sticky dark-colored substance produced from the collected buds or exudates of plants (resin) by bees (Apis mellifera L.). The resin is masticated, salivary enzymes are added, and the partially digested material is mixed with beewax and used in the hive to seal the walls, strengthen the borders of combs, and embalm dead invaders (Wollenweber et al., 1990). The propolis samples are from NMR data and were collected in the autumn (AU), winter (WI), spring (SP), and summer (SM) of 2010 from Apis mellifera hives located in southern Brazil (Santa Catarina State). A total of 59 samples were collected, and the distribution of samples by seasons being: SM - 16 samples, AU and SP - 15 samples, WI - 13 samples. Also, three agroecological regions were defined for the different apiaries, and one distributed as follows: Highlands - 12 samples, Plain - 11 samples, Plateau - 36 samples.

Usage

```
data(propolisSampleList)
```

Format

An object of class `"list"`

References

E. Wollenweber, B. M. Hausen, and W. Greenaway. Phenolic constituents and sensitizing properties of propolis, poplar balsam and balsam of peru. Bulletin de Groupe Polyphenol, 15:112-120, 1990. M. Maraschin, A. Somensi-Zeggio, S. K. Oliveira, S. Kuhnen, M. M. Tomazzoli, A. C. M. Zeri, R. Carreira, and M. Rocha. A machine learning and chemometrics assisted interpretation of spectroscopic data - a nmr-based metabolomics platform for the assessment of brazilian propolis. 2012

Examples

```
data(propolisSampleList)
propolisSampleList[[1]]
```

read_csvs_folder *Read CSVs from folder*

Description

Reads multiple CSV files in a given folder.

Usage

```
read_csvs_folder(foldername, ...)
```

Arguments

foldername	string with the name of the folder.
...	additional parameters to read.csv function.

Value

Returns a list of data frames.

Examples

```
## Not run:
    ## Example of reading multiple csvs
    sample.list = read_csvs_folder("foldername")

## End(Not run)
```

read_dataset_csv *Read dataset from CSV*

Description

Reads the data from a CSV file and creates the dataset.

Usage

```
read_dataset_csv(filename.data, filename.meta = NULL,
type = "undefined", description = "", label.x = NULL,
label.values = NULL, sample.names = NULL, format = "row",
header.col = TRUE, header.row = TRUE, sep = ",",
header.col.meta = TRUE, header.row.meta = TRUE, sep.meta = ",")
```

Arguments

filename.data	name of the data file.
filename.meta	name of the metadata file.
type	type of the data.
description	a short text describing the dataset.
label.x	the label for the x values.
label.values	the label for the y values.
sample.names	the names of the samples.
format	format which the data are in the CSV file. It can be "row" if the samples are in the rows or "col" if the samples are in the columns.
header.col	boolean value indicating if the CSV contains a header column with the names of the samples or variables.
header.row	boolean value indicating if the CSV contains a header row with the names of the samples or variables.
sep	the separator character.
header.col.meta	boolean value indicating if the metadata CSV file contains a header column with the name of the metadata variables.
header.row.meta	boolean value indicating if the metadata CSV file contains a header row with the name of the samples.
sep.meta	the separator character of the metadata file.

Value

Returns the dataset from the CSV file.

Examples

```
## Not run:
  ## Example of reading a dataset from a CSV file
  dataset = read_dataset_csv("data.csv", "metadata.csv",
    type = "nmr-spectra",
    description ="description of the dataset",
    label.x = "ppm", label.values = "intensity")

## End(Not run)
```

read_dataset_dx *Read dataset from (J)DX files*

Description

Reads the data from the (J)DX files and creates the dataset.

Usage

```
read_dataset_dx(folder.data, filename.meta = NULL,
type = "undefined", description = "", label.x = NULL,
label.values = NULL, header.col.meta = TRUE,
header.row.meta = TRUE, sep.meta = ",")
```

Arguments

folder.data	string containing the path of the data folder.
filename.meta	name of the metadata file.
type	type of the data.
description	a short text describing the dataset.
label.x	the label for the x values.
label.values	the label for the y values.
header.col.meta	boolean value indicating if the metadata CSV file contains a header column with the name of the metadata variables.
header.row.meta	boolean value indicating if the metadata CSV file contains a header row with the name of the samples.
sep.meta	the separator character of the metadata file.

Value

Returns the dataset from the (J)DX files.

Examples

```
## Not run:
  ## Example of reading a dataset from (J)DX files
  dataset = read_dataset_dx("data", "metadata.csv", type = "nmr-spectra",
    description ="description of the dataset", label.x = "ppm",
    label.values = "intensity")

## End(Not run)
```

read_dataset_spc *Read dataset from SPC files*

Description

Reads the data from the SPC files and creates the dataset.

Usage

```
read_dataset_spc(folder.data, filename.meta = NULL,
type = "undefined", description = "", nosubhdr = F,
label.x = NULL, label.values = NULL, header.col.meta = TRUE,
header.row.meta = TRUE, sep.meta = ",")
```

Arguments

folder.data	string containing the path of the data folder.
filename.meta	name of the metadata file.
type	type of the data.
description	a short text describing the dataset.
nosubhdr	if TRUE, it won't read the subheader.
label.x	the label for the x values.
label.values	the label for the y values.
header.col.meta	boolean value indicating if the metadata CSV file contains a header column with the name of the metadata variables.
header.row.meta	boolean value indicating if the metadata CSV file contains a header row with the name of the samples.
sep.meta	the separator character of the metadata file.

Value

Returns the dataset from the SPC files.

Examples

```
## Not run:
  ## Example of reading a dataset from SPC files
  dataset = read_dataset_spc("data", "metadata.csv", type = "ir-spectra",
    description = "description of the dataset", label.x = "wavelength",
    label.values = "transmittance")

## End(Not run)
```

read_data_csv *Read CSV data*

Description

Reads the data from the CSV file.

Usage

```
read_data_csv(filename, format = "row", header.col = TRUE,
header.row = TRUE, sep = ",")
```

Arguments

filename	name of the file with the data.
format	format which the data are in the CSV file. It can be "row" if the samples are in the rows or "col" if the samples are in the columns.
header.col	boolean value indicating if the CSV contains a header column with the names of the samples or variables.
header.row	boolean value indicating if the CSV contains a header row with the names of the samples or variables.
sep	the separator character.

Value

Returns a numeric matrix with the data.

Examples

```
## Not run:
  ## Example of reading a dataset from a CSV file
  data.matrix = read_data_csv("data.csv", format = "row")

## End(Not run)
```

read_data_dx *Read data from (J)DX files*

Description

Reads the data from the (J)DX files.

Usage

```
read_data_dx(foldername, debug = F)
```

Arguments

foldername	string with the path of the data folder.
debug	debug option for ChemoSpec's readJDX function.

Value

Returns a list with the samples and the respective data points.

Examples

```
## Not run:
  ## Example of reading a dataset from (J)DX file
s  sample.list = read_data_dx("data")

## End(Not run)
```

---

read_data_spc *Read data from SPC files*

Description

Reads the data from the SPC files.

Usage

```
read_data_spc(foldername, nosubhdr = F)
```

Arguments

foldername	string with the path of the data folder.
nosubhdr	if TRUE, it won't read the subheader.

Value

Returns a list with the samples and the respective data points.

Examples

```
## Not run:
  ## Example of reading a dataset from SPC files
  sample.list = read_data_spc("data")

## End(Not run)
```

read_metadata *Read metadata*

Description

Read the metadata from a file.

Usage

```
read_metadata(filename, header.col = T, header.row = T,
sep = ",")
```

Arguments

filename	name of the file with the metadata.
header.col	boolean value indicating if the metadata CSV file contains a header column with the name of the metadata variables.
header.row	boolean value indicating if the metadata CSV file contains a header row with the name of the samples.
sep	the separator character.

Value

Returns a data frame with the metadata.

Examples

```
## Not run:
  ## Example of reading metadata from a file
  metadata = read_metadata("metadata.csv")

## End(Not run)
```

read_ms_spectra *Read MS spectra*

Description

Read the data from the MS files and creates the dataset.

Usage

```
read_ms_spectra(folder.name, type = "undefined",
filename.meta = NULL, description = "", prof.method = "bin",
fwhm = 30, bw = 30, intvalue = "into", header.col.meta = TRUE,
header.row.meta = TRUE, sep.meta = ",")
```

Arguments

folder.name	string containing the path of the data folder.
type	type of the data.
filename.meta	name of the metadata file.
description	a short text describing the dataset.
prof.method	profmethod parameter from xcmsSet function from xcms package.
fwhm	fwhm parameter from xcmsSet function from xcms package. A commonly used value is 30 (seconds) for LC-MS and 4 (seconds) for GC-MS spectra.
bw	bw parameter from group function from xcms package.
intvalue	value parameter from groupval function from xcms package. It can be: • **"into"** - integrated area of original (raw) peak • **"intf"** - integrated area of filtered peak. • **"maxo"** - maximum intensity of original (raw) peak. • **"maxf"** - maximum intensity of filtered peak.
header.col.meta	boolean value indicating if the metadata CSV file contains a header column with the name of the metadata variables.
header.row.meta	boolean value indicating if the metadata CSV file contains a header row with the name of the samples.
sep.meta	the separator character of the metadata file.

Value

Returns a dataset from the MS files.

Examples

```
## Not run:
  ## Example of reading a dataset from MS files
  dataset = read_ms_spectra("data", type = "nmr-spectra", "metadata.csv",
    description = "description of the dataset")

## End(Not run)
```

---

read_multiple_csvs *Read multiple CSVs*

Description

Reads multiple CSVs, each one with a sample.

Usage

```
read_multiple_csvs(filenames, ext = ".csv", ...)
```

Arguments

filenames	list of file names of the files to read.
ext	extension name.
...	additional parameters to read.csv function.

Value

returns a list of dataframes.

Examples

```
## Not run:
    ## Example of reading multiple csvs
    filenames = c("sample1", "sample2", "sample3", "sample4")
    sample.list = read_multiple_csvs(filenames, ext = ".csv")

## End(Not run)
```

---

recursive_feature_elimination

Perform recursive feature elimination

Description

Perform recursive feature elimination on the dataset using caret's package.

Usage

```
recursive_feature_elimination(datamat, samples.class,
functions = caret::rfFuncs, method = "cv", repeats = 5,
number = 10, subsets = 2^(2:4))
```

Arguments

datamat	data matrix from dataset.
samples.class	string or index indicating what metadata to use.
functions	a list of functions for model fitting, prediction and variable importance.
method	the external resampling method: boot, cv, LOOCV or LGOCV (for repeated training/test splits.
repeats	for repeated k-fold cross-validation only: the number of complete sets of folds to compute.
number	either the number of folds or number of resampling iterations.
subsets	a numeric vector of integers corresponding to the number of features that should be retained.

Value

A caret's rfe object with the result of recursive feature selection.

Examples

```
## Not run:
  ## Example of recursive feature elimination
  data(cachexia)
  library(caret)
  rfe.result = recursive_feature_elimination(cachexia$data,
       cachexia$metadata$Muscle.loss, functions = caret::rfFuncs,
       method = "cv", number = 3, subsets = 2^(1:6))

## End(Not run)
```

`remove_data` *Remove data*

Description

Remove data from the dataset.

Usage

```
remove_data(dataset, data.to.remove, type = "sample",
by.index = F, rebuild.factors = T)
```

Arguments

`dataset`	list representing the dataset from a metabolomics experiment.
`data.to.remove`	vector with the elements' names to remove
`type`	type of the element to remove. It can be: • **"sample"** to remove samples. • **"data"** to remove variables. • **"metadata"** to remove metadata's variables.
`by.index`	if TRUE, the values of the data.to.remove argument are indexes in case the type is "data".
`rebuild.factors`	if TRUE, rebuilds the factors from metadata.

Value

Returns the dataset with the specified data removed.

Examples

```
## Example of removing data
data(cachexia)
dataset = remove_data(cachexia, c("Creatine","Serine"), type = "data",
  by.index = FALSE)
```

`remove_data_variables` *Remove data variables*

Description

Remove data variables from the dataset.

Usage

```
remove_data_variables(dataset, variables.to.remove,
by.index = FALSE)
```

Arguments

`dataset`	list representing the dataset from a metabolomics experiment.
`variables.to.remove`	vector with the variables' names to remove.
`by.index`	if TRUE, the values of the variables.to.remove argument are indexes.

Value

Returns the dataset with the specified data variables removed.

Examples

```
## Example of removing data variables
data(cachexia)
dataset = remove_data_variables(cachexia, c("Creatine","Serine"),
  by.index = FALSE)
```

remove_metadata_variables *Remove metadata's variables*

Description

Remove metadata's variables from the dataset

Usage

```
remove_metadata_variables(dataset, variables.to.remove)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
variables.to.remove	vector with the metadata's variables to remove.

Value

Returns the dataset with the selected metadata's variables removed.

Examples

```
## Example of removing metadata's variables
data(cassavaPPD)
dataset = remove_metadata_variables(cassavaPPD, c("varieties"))
```

remove_peaks_interval *Remove interval of peaks*

Description

Removes peaks from a given interval.

Usage

```
remove_peaks_interval(sample.df, peak.val.min, peak.val.max)
```

Arguments

sample.df	data frame with the samples' peaks.
peak.val.min	minimum peak value.
peak.val.max	maximum peak value.

Value

Returns a data frame with the filtered samples' peaks.

Examples

```
## Example of removing a interval of peaks
data(propolisSampleList)
samples.df = remove_peaks_interval(propolisSampleList[[1]], 2, 8.05)
```

`remove_peaks_interval_sample_list` *Remove interval of peaks (sample list)*

Description

Removes peaks on a sample list given a peak interval.

Usage

```
remove_peaks_interval_sample_list(sample.list, peak.val.min,
peak.val.max)
```

Arguments

`sample.list`	list of data frames with the samples' peaks.
`peak.val.min`	minimum peak value.
`peak.val.max`	maximum peak value.

Value

Returns the sample list with the filtered peaks.

Examples

```
## Example of removing a interval of peaks in a sample list
data(propolisSampleList)
samples.list = remove_peaks_interval_sample_list(propolisSampleList, 2, 8.05)
```

remove_samples *Remove samples*

Description

Remove samples from the dataset.

Usage

```
remove_samples(dataset, samples.to.remove, rebuild.factors = T)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
samples.to.remove	vector with the sample's names to remove.
rebuild.factors	if TRUE, rebuilds the factors from metadata.

Value

Returns the dataset with the specified samples removed.

Examples

```
## Example of removing samples
data(cachexia)
cachexia = remove_samples(cachexia, c("PIF_178","PIF_090"))
```

remove_samples_by_nas *Remove samples by NAs*

Description

Remove samples from the dataset by the number of NAs

Usage

```
remove_samples_by_nas(dataset, max.nas = 0, by.percent = F)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
max.nas	number of NAs (or percentage) to which samples with more NAs will be removed.
by.percent	if TRUE the max.nas argument is a percentage.

Value

Returns the dataset with the samples with more NAs than the limit removed.

Examples

```
## Example of removing samples by NAs
data(propolis)
propolis = remove_samples_by_nas(propolis, 40, by.percent = TRUE)
```

`remove_samples_by_na_metadata`

Remove samples by NA on metadata

Description

Remove samples from the dataset with the metadata's variable value with NAs.

Usage

```
remove_samples_by_na_metadata(dataset, metadata.var)
```

Arguments

`dataset`	list representing the dataset from a metabolomics experiment.
`metadata.var`	metadata's variable.

Value

Returns the dataset with the samples with NA on metadata's variable removed.

Examples

```
## Example of removing samples that have NAs on metadata's variable
data(cachexia)
cachexia$metadata$Muscle.loss[10] = NA
cachexia = remove_samples_by_na_metadata(cachexia, "Muscle.loss")
```

remove_variables_by_nas

Remove variables by NAs

Description

Remove variables from the dataset by the number of NAs

Usage

```
remove_variables_by_nas(dataset, max.nas = 0, by.percent = F)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
max.nas	number of NAs (or percentage) to which samples with more NAs will be removed.
by.percent	if TRUE the max.nas argument is a percentage.

Value

Returns the dataset with the variables with more NAs than the limit removed.

Examples

```
## Example of removing variables by NAs
data(propolis)
propolis = remove_variables_by_nas(propolis, 40, by.percent = TRUE)
```

remove_x_values_by_interval

Remove x-values by interval

Description

Remove an interval of x-values from the dataset.

Usage

```
remove_x_values_by_interval(dataset, min.value, max.value)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
min.value	minimum value of the interval.
max.value	maximum value of the interval.

Value

Returns the dataset with the specified interval of x-values removed.

Examples

```
## Example of removing x-values by interval
data(cassavaPPD)
cassavaPPD = remove_x_values_by_interval(cassavaPPD, 200, 300)
```

`replace_data_value` *Replace data value*

Description

Replace a data value for a new value on the dataset.

Usage

```
replace_data_value(dataset, x.axis.val, sample, new.value,
by.index = F)
```

Arguments

`dataset`	list representing the dataset from a metabolomics experiment.
`x.axis.val`	variable index or name.
`sample`	sample name.
`new.value`	new value to replace the old value.
`by.index`	boolean value to indicate if the x.axis.val is an index or not.

Value

Returns the dataset with the data value replaced.

Examples

```
## Example of replacing a data value from the dataset
data(cachexia)
dataset = replace_data_value(cachexia, "Creatine", "PIF_178", 10.3,
  by.index = FALSE)
```

replace_metadata_value *Replace metadata's value*

Description

Replace a metadata's variable value of a sample.

Usage

```
replace_metadata_value(dataset, variable, sample, new.value)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
variable	metadata's variable.
sample	name of the sample.
new.value	new value of the metadata's variable.

Value

Returns the dataset with the metadata updated.

Examples

```
## Example of replacing metadata's variable value of a sample
data(cachexia)
dataset = replace_metadata_value(cachexia, "Muscle.loss", "PIF_178",
  "control")
```

savitzky_golay *Savitzky-golay transformation*

Description

Smoothing and derivative of the data using Savitzky-Golay.

Usage

```
savitzky_golay(dataset, p.order, window, deriv = 0)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
p.order	integer value representing the polynomial order.
window	odd number indicating the window size.
deriv	integer value indicating the differentiation order.

Value

Returns the dataset with the spectra smoothed using Savitzky-Golay.

Examples

```
## Example of smoothing the spectra from dataset with Savitzky-Golay
data(cassavaPPD)
dataset.smoothed = savitzky_golay(cassavaPPD, p.order = 3, window = 11,
   deriv = 0)
```

---

scaling *Scale dataset*

Description

Performs scaling according to a method.

Usage

```
scaling(dataset, method = "auto")
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
method	string specifying the scaling method. The possible values are: • **"auto"** auto scaling. • **"range"** range scaling. • **"pareto"** pareto scaling. • **"tointerval"** scaling to an interval.

Value

Returns the dataset scaled.

Examples

```
## Example of auto scaling
data(cassavaPPD)
cassava.scal = scaling(cassavaPPD, "auto")
```

scaling_samples *Scale data matrix*

Description

Performs scaling according to a method.

Usage

```
scaling_samples(datamat, method = "auto")
```

Arguments

datamat	data matrix.
method	string specifying the scaling method. The possible values are: • **"auto"** auto scaling. • **"range"** range scaling. • **"pareto"** pareto scaling. • **"tointerval"** scaling to an interval.

Value

Returns the data matrix scaled.

Examples

```
## Example of auto scaling
data(cassavaPPD)
cassava.scal = scaling_samples(cassavaPPD$data, "auto")
```

set_metadata *Set new metadata*

Description

Updates the dataset's metadata with a new one.

Usage

```
set_metadata(dataset, new.metadata)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
new.metadata	matrix or dataframe with the new metadata.

Value

Returns the dataset with the updated metadata.

Examples

```
## Example of setting a new metadata to the dataset
data(cachexia)
new.metadata = c(rep("meta1", 39), rep("meta2", 38))
new.metadata = data.frame(var_meta = new.metadata)
rownames(new.metadata) = get_sample_names(cachexia)
cachexia = set_metadata(cachexia, new.metadata)
```

---

set_sample_names *Set samples names*

Description

Set new samples names to the dataset.

Usage

```
set_sample_names(dataset, new.sample.names)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
new.sample.names	vector with the new samples names.

Value

Returns the dataset with the updated samples names.

Examples

```
## Example of setting a new value label to the dataset
data(cachexia)
new.samples.names = as.character(1:77)
cachexia = set_sample_names(cachexia, new.samples.names)
```

set_value_label *Set value label*

Description

Set a new value label for the dataset.

Usage

```
set_value_label(dataset, new.val.label)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
new.val.label	string with the new value label.

Value

Returns the dataset with the updated value label.

Examples

```
## Example of setting a new value label to the dataset
data(cachexia)
cachexia = set_value_label(cachexia, "new value label")
```

set_x_label *Set x-label*

Description

Set a new x-label to the dataset.

Usage

```
set_x_label(dataset, new.x.label)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
new.x.label	string with the x-label.

Value

Returns the dataset with the updated x-label.

Examples

```
## Example of setting a new x-label to the dataset
data(cachexia)
cachexia = set_x_label(cachexia, "new x-label")
```

---

set_x_values *Set new x-values*

Description

Set new x-values to the dataset

Usage

```
set_x_values(dataset, new.x.values, new.x.label = NULL)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
new.x.values	vector with the new x-values.
new.x.label	string with the new x-label (can be NULL).

Value

Returns the dataset with the updated x-values.

Examples

```
## Example of setting new x-values to the dataset
data(cachexia)
new.xvalues = 1:63
cachexia = set_x_values(cachexia, new.xvalues, new.x.label = NULL)
```

---

shift_correction *Shift correction*

Description

Shifts the spectra according to a specific method.

Usage

```
shift_correction(dataset, method = "constant", shift.val = 0,
interp.function = "linear", ref.limits = NULL)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
method	string that indicates the shifting method. It can be: • "constant" uses a constant shift that is added to the x-values • "interpolation" uses interpolation according to "interp.function"
shift.val	value of the shift (for constant and interpolation methods); can be a single value for all spectra, can be the string "auto", the shifts are automatically determined or a vector with the size of the number of samples with the shifts for each spectra.
interp.function	string that represents the interpolation function, can be "linear" or "spline".
ref.limits	vector with 2 elements that represents the reference limits to calculate the shifts.

Value

Returns the dataset with the spectras shifted.

Examples

```
## Example of shift correction with linear interpolation and
## shifts calculated based on a reference band of the spectra
data(cassavaPPD)
dataset.corrected = shift_correction(cassavaPPD, method = "interpolation",
    shift.val = "auto", interp.function = "linear",
    ref.limits = c(800,850))
```

---

smoothing_interpolation

Smoothing interpolation

Description

Performs smoothing interpolation according to a specific method.

Usage

```
smoothing_interpolation(dataset, method = "bin",
reducing.factor = 2, x.axis = NULL, p.order = 3,
window = 11, deriv = 0)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
method	string specifying the smoothing method. The three possible methods are: "bin", "loess" and "savitzky.golay".
reducing.factor	numeric value indicating the reducing factor for bin method.

x.axis	numeric vector representing the new x-axis for loess method.
p.order	numeric value representing the polynomial order for savitzky-golay method.
window	numeric value indicating the size of the window for savitzky-golay method. (must be an odd number)
deriv	numeric value indicating the differentiation order for savitzky-golay method.

Value

Returns the dataset with the spectra smoothed.

Examples

```
## Example of smoothing the spectra from dataset
data(cassavaPPD)
dataset.smoothed = smoothing_interpolation(cassavaPPD, method = "bin",
   reducing.factor = 2)
```

---

snv_dataset *Standard Normal Variate*

Description

Performs Standard Normal Variate on the dataset.

Usage

```
snv_dataset(dataset)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.

Value

Returns the dataset with the data normalized by SNV.

Examples

```
## Example of SNV on a dataset
data(cassavaPPD)
dataset = snv_dataset(cassavaPPD)
```

spinalCord — *Brazilian Propolis from different Harvest Seasons and different Agroecological Regions*

Description

This dataset consists of 12 LC-MS samples of spectra in the netCDF format, from mice spinal cord divided into 2 groups: the wild type and the knockout group. The data was obtained from the MetaboAnalyst site, originating from a study which describes a general strategy for identifying endogenous substrates of enzymes by untargeted LC-MS analysis of tissue metabolomes from wild-type and enzyme-inactivated organisms

Usage

```
data(spinalCord)
```

Format

An object of class "list"

Source

MetaboAnalyst

References

A. Saghatelian, S.A. Trauger, E.J. Want, E.G. Hawkins, G. Siuzdak, B.F. Cravatt Assignment of Endogenous Substrates to Enzymes by Global Metabolite Profiling Biochemistry, 43:14332-14339, 2004.

Examples

```
data(spinalCord)
sum_dataset(spinalCord)
```

stats_by_sample — *Statistics of samples*

Description

Get a summary of statistics of the samples.

Usage

```
stats_by_sample(dataset, samples = NULL)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
samples	if defined restricts the application to a given set of samples.

Value

Returns a vector with the a summary of statistics of the samples.

Examples

```
## Example of getting stats of samples
data(cachexia)
samples.stats.result = stats_by_sample(cachexia)
```

---

stats_by_variable *Statistics of variables*

Description

Get a summary of statistics of the variables.

Usage

```
stats_by_variable(dataset, variables = NULL,
variable.bounds = NULL)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
variables	allows to define which variables to calculate the stats (if numbers, indexes are assumed).
variable.bounds	allow to define an interval of variables (if numeric).

Value

Returns a vector with the a summary of statistics of the variables.

Examples

```
## Example of getting stats of variables
data(cachexia)
variable.stats.result = stats_by_variable(cachexia)
```

subset_by_samples_and_xvalues *Subset by samples and x-values*

Description

Gets a subset of specific samples and x-values.

Usage

```
subset_by_samples_and_xvalues(dataset, samples, variables = NULL,
by.index = F, variable.bounds = NULL, rebuild.factors = T)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
samples	vector with indexes or names of the samples to select
variables	vector with the desired variables to get from the dataset.
by.index	if TRUE, the values of the variables argument are indexes.
variable.bounds	variable bounds used if by.index is FALSE and variables are NULL.
rebuild.factors	if TRUE the metadata factors are rebuilded.

Value

Returns the dataset with the selected samples and x-values.

Examples

```
## Example of subsetting samples and x-values
data(cachexia)
subset = subset_by_samples_and_xvalues(cachexia, c("PIF_178","NETL_022_V1"),
 variables = c("Creatine","Serine"))
```

subset_metadata *Subset metadata*

Description

Subsets the metadata according to the specified metadata's variables.

Usage

```
subset_metadata(dataset, variables)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
variables	metadata's variables.

Value

Returns the dataset with the metadata subsetted.

Examples

```
## Example of subsetting samples
data(propolis)
subset = subset_metadata(propolis, c("seasons"))
```

---

subset_random_samples *Subset random samples*

Description

Gets a subset of random samples from the dataset.

Usage

```
subset_random_samples(dataset, nsamples)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
nsamples	integer representing the number of samples that we want to get.

Value

Returns the dataset with a number of random samples.

Examples

```
## Example of subsetting random samples
data(cachexia)
subset = subset_random_samples(cachexia, 15)
```

subset_samples *Subset samples*

Description

Gets a subset of specific samples from the dataset.

Usage

```
subset_samples(dataset, samples, rebuild.factors = T)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
samples	vector with indexes or names of the samples to select
rebuild.factors	if TRUE the metadata factors are rebuilded.

Value

Returns the dataset with the selected set of samples.

Examples

```
## Example of subsetting samples
data(cachexia)
subset = subset_samples(cachexia, c("PIF_178","PIF_132"))
```

subset_samples_by_metadata_values *Subset samples by metadata values*

Description

Gets a subset of specific samples according to metadata's values from the dataset.

Usage

```
subset_samples_by_metadata_values(dataset, metadata.varname,
values)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
metadata.varname	name of the metadata's variable.
values	values of the metadata's variable.

Value

Returns the dataset with the set of samples according to the metadata's values.

Examples

```
## Example of subsetting samples by metadata's values
data(cassavaPPD)
subset = subset_samples_by_metadata_values(cassavaPPD, "varieties",
 c("BRA","IAC"))
```

`subset_x_values` *Subset x-values*

Description

Gets a subset of specific x-values from the dataset.

Usage

```
subset_x_values(dataset, variables, by.index = FALSE)
```

Arguments

`dataset`	list representing the dataset from a metabolomics experiment.
`variables`	vector with the desired variables to get from the dataset.
`by.index`	if TRUE, the values of the variables argument are indexes.

Value

Returns the dataset with the selected set of x-values.

Examples

```
## Example of subsetting x-values
data(cachexia)
subset = subset_x_values(cachexia, c(1,2,10,20), by.index = TRUE)
```

subset_x_values_by_interval

Subset x-values by interval

Description

Gets a subset of a specific interval of x-values.

Usage

```
subset_x_values_by_interval(dataset, min.value, max.value)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
min.value	the minimum value of the interval.
max.value	the maximum value of the interval.

Value

Returns the dataset with the selected interval of x-values.

Examples

```
## Example of subsetting x-values by an interval
data(cassavaPPD)
subset = subset_x_values_by_interval(cassavaPPD, 200, 800)
```

summary_var_importance

Summary of variables importance

Description

Summary of variables importance of the models

Usage

```
summary_var_importance(performances, number.rows)
```

Arguments

performances	the result from training the models.
number.rows	number of variables to display.

Value

Returns list with the variables importance of each model.

Examples

```
  ## Example of getting a summary of variables importance
  data(cachexia)
  training.result = train_models_performance(cachexia, "pls",
"Muscle.loss", "cv")
  result = summary_var_importance(training.result, 10)
```

`sum_dataset` *Dataset summary*

Description

Returns a summary with its main features.

Usage

```
sum_dataset(dataset, stats = T)
```

Arguments

`dataset`	list representing the dataset from a metabolomics experiment.
`stats`	if TRUE prints some global statistics of the data values.

Value

Returns the summary of the dataset containing:

- Description
- Type of data
- Number of samples
- Number of datapoints
- Number of metadata variables if metadata not null
- Labels of x axis values and data points if labels not null

If the parameter 'stats' is TRUE then returns also:

- Number of missing values in data
- Mean of data values
- Median of data values
- Standard deviation
- Range of values
- Quantiles

Examples

`train_and_predict` *Train and predict*

Description

Train a model and predict new unlabeled samples with that model.

Usage

```
train_and_predict(dataset, new.samples, column.class, model,
validation, num.folds = 10, num.repeats = 10, tunelength = 10,
tunegrid = NULL, metric = NULL, summary.function =
defaultSummary)
```

Arguments

`dataset`	list representing the dataset from a metabolomics experiment.
`new.samples`	dataframe with new samples to predict the class label.
`column.class`	metadata column class.
`model`	model to be used in training.
`validation`	validation method.
`num.folds`	number of folds in cross validation.
`num.repeats`	number of repeats.
`tunelength`	number of levels for each tuning parameters.
`tunegrid`	dataframe with possible tuning values.
`metric`	metric used to evaluate the model's performance. Can be "Accuracy" or "ROC".
`summary.function`	summary function. For "ROC" the multiClassSummary function must be used.

Value

Returns a list with the training result and the predictions result.

Examples

```
## Example of training and predicting
data(cachexia)
result = train_and_predict(cachexia, new.samples = cachexia$data,
"Muscle.loss", "pls", "cv")
```

`train_classifier` *Train classifier*

Description

Train a specific classifier.

Usage

```
train_classifier(dataset, column.class, model, validation,
num.folds = 10, num.repeats = 10, tunelength = 10,
tunegrid = NULL, metric = NULL,
summary.function = defaultSummary, class.in.metadata = T)
```

Arguments

`dataset`	list representing the dataset from a metabolomics experiment.
`column.class`	metadata column class.
`model`	model to be used in training.
`validation`	validation method.
`num.folds`	number of folds in cross validation.
`num.repeats`	number of repeats.
`tunelength`	number of levels for each tuning parameters.
`tunegrid`	dataframe with possible tuning values.
`metric`	metric used to evaluate the model's performance. Can be "Accuracy" or "ROC".
`summary.function`	summary function. For "ROC" the multiClassSummary function must be used.
`class.in.metadata`	boolean value to indicate if the class is in metadata.

Value

Returns the train result object from caret.

Examples

```
## Example of training a classifier
data(cachexia)
result = train_classifier(cachexia, "Muscle.loss", "pls", "cv")
```

`train_models_performance` *Train models*

Description

Train various models.

Usage

```
train_models_performance(dataset, models, column.class,
validation, num.folds = 10, num.repeats = 10, tunelength = 10,
tunegrid = NULL, metric = NULL, summary.function = "default",
class.in.metadata = T, compute.varimp = T)
```

Arguments

`dataset`	list representing the dataset from a metabolomics experiment.
`models`	models to be used in training.
`column.class`	metadata column class.
`validation`	validation method.
`num.folds`	number of folds in cross validation.
`num.repeats`	number of repeats.
`tunelength`	number of levels for each tuning parameters.
`tunegrid`	dataframe with possible tuning values.
`metric`	metric used to evaluate the model's performance. Can be "Accuracy" or "ROC".
`summary.function`	summary function. For "ROC" the multiClassSummary function must be used.
`class.in.metadata`	boolean value to indicate if the class is in metadata.
`compute.varimp`	boolean value to indicate if the var importance is calculated.

Value

Returns a list with the results from training

`performance`	The results from the best tunes of the models
`vips`	The variable importance from the models
`full.results`	The full results from the tuning parameters of each model
`best.tunes`	The best tune of each model
`confusion.matrices`	The confusion matrices of the models (only in classification)
`final.models`	The final models

Examples

```
## Example of training models
data(cachexia)
result = train_models_performance(cachexia, "pls",
 "Muscle.loss", "cv")
```

`transform_data` *Transform data*

Description

Performs data transformation according to a method.

Usage

```
transform_data(dataset, method = "log")
```

Arguments

`dataset`	list representing the dataset from a metabolomics experiment.
`method`	string specifying the transformation method. The possible values are: • **"log"** logarithmic transformation. • **"cubicroot"** cubic root transformation.

Value

Returns the dataset with the data transformation applied.

Examples

```
## Example of logarithmic transformation
data(cachexia)
dataset.log = transform_data(cachexia, "log")
```

transmittance_to_absorbance *Convert transmittance to absorbance*

Description

Converts transmittance values to absorbance values.

Usage

```
transmittance_to_absorbance(dataset, percent = T)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
percent	boolean value to indicate if the absorbance values are in a percentage or not.

Value

Returns the dataset with the data points converted to absorbance values.

Examples

```
## Example of converting transmittance values to absorbance values
data(cassavaPPD)
dataset = transmittance_to_absorbance(cassavaPPD, percent = TRUE)
```

tTests_dataset *t-Tests on dataset*

Description

Run t-Tests for each row of the data from the dataset.

Usage

```
tTests_dataset(dataset, metadata.var, threshold = NULL,
write.file = F, file.out = "ttests.csv")
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
metadata.var	metadata variable to use in the t-tests.
threshold	threshold value of the p-value.
write.file	boolean value to write or not a file with the results.
file.out	name of the file.

Value

Table with the results of the t-tests, with p-value, -log10(p-value) and false discovery rate (fdr).

Examples

```
## Example of t-Tests on dataset
data(cachexia)
ttests.result = tTests_dataset(cachexia, "Muscle.loss",
write.file = FALSE)
```

`values_per_peak` *Values per peak*

Description

Gets the number of values on each peak.

Usage

```
values_per_peak(samples.df)
```

Arguments

`samples.df`	data frame with the samples' peaks.

Value

Returns a vector with the number of values for each peak.

Examples

```
## Example of getting the number of values for each peak
data(propolis)
num.peaks = values_per_peak(propolis$data)
```

values_per_sample *Values per peak*

Description

Gets the number of values on each sample.

Usage

```
values_per_sample(samples.df)
```

Arguments

samples.df	data frame with the samples' peaks.

Value

Returns a vector with the number of values for each sample.

Examples

```
## Example of getting the number of values for each sample
data(propolis)
num.values.samples = values_per_sample(propolis$data)
```

variables_as_metadata *Variables as metadata*

Description

Use one or more data variables as metadata variables.

Usage

```
variables_as_metadata(dataset, variables, by.index = F)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
variables	name or index of the variables that are going to be used as metadata variables.
by.index	boolean value indicating if the variables are indexes or names

Value

Returns the dataset with the variables removed from the data and added on the metadata.

Examples

```
## Example of using a variable as metadata variable
data(cachexia)
dataset = variables_as_metadata(cachexia, "Creatine", by.index = FALSE)
```

`volcano_plot_fc_tt` *Volcano plot*

Description

Volcano plot to intersect the results from t-tests and fold change.

Usage

```
volcano_plot_fc_tt(dataset, fc.results, tt.results,
fc.threshold = 2, tt.threshold = 0.01)
```

Arguments

`dataset`	list representing the dataset from a metabolomics experiment.
`fc.results`	fold change results.
`tt.results`	t-tests results.
`fc.threshold`	fold change threshold value.
`tt.threshold`	t-test p-value threshold.

Value

Returns the name of the samples which intersects both fold change and t-tests results above the given thresholds.

Examples

```
## Example of a volcano plot
data(cachexia)
foldchange.results = fold_change(cachexia, "Muscle.loss", "control")
ttests.results = tTests_dataset(cachexia, "Muscle.loss")
volcano_plot_fc_tt(cachexia, foldchange.results, ttests.results,
   fc.threshold = 2, tt.threshold = 0.01)
```

xvalue_interval_to_indexes *Get indexes of an interval of x-values*

Description

Returns indexes corresponding to an interval of x-values (only to numerical values - spectra)

Usage

```
xvalue_interval_to_indexes(dataset, min.value, max.value)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
min.value	mininum x-value of the interval.
max.value	maximum x-value of the interval.

Value

Returns a numeric vector with the indexes of the x-values interval

Examples

```
## Example of getting the indexes of an interval of x-values
data(propolis)
indexes.interval = xvalue_interval_to_indexes(propolis, 2.0, 5.5)
```

x_values_to_indexes *Get x-values indexes*

Description

Returns the indexes corresponding to a vector of x-values (only to numerical values - spectra)

Usage

```
x_values_to_indexes(dataset, x.values)
```

Arguments

dataset	list representing the dataset from a metabolomics experiment.
x.values	vector of x-values.

Value

Returns a numeric vector with the indexes of the x-values.

Examples

```
## Example of getting the indexes of the x-values
data(propolis)
indexes = x_values_to_indexes(propolis, c(1.3, 3.51, 6.03))
```

Index