Package 'fdm2id'

June 12, 2023

Title Data Mining and R Programming for Beginners

Version 0.9.9

Description
Contains functions to simplify the use of data mining methods (classification, regression, clustering, etc.), for students and beginners in R programming. Various R packages are used and wrappers are built around the main functions, to standardize the use of data mining methods (input/output): it brings a certain loss of flexibility, but also a gain of simplicity. The package name came from the French ``Fouille de Données en Master 2 Informatique Décisionnelle".

Depends R (>= 3.5.0), arules, arulesViz, FactoMineR

Imports mclust, methods, nnet, pls

Suggests car, caret, class, cluster, datasets, e1071, fds, flexclust, fpc, glmnet, graphics, grDevices, ibr, irr, kohonen, leaps, MASS, mda, meanShiftR, questionr, randomForest, ROCR, rpart, rpart.plot, Rtsne, SnowballC, stats, text2vec, stopwords, utils, wordcloud, xgboost

Enhances NMF

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 7.2.3

NeedsCompilation no

Author Alexandre Blansché [aut, cre]

Maintainer Alexandre Blansché <alexandre.blansche@univ-lorraine.fr>

Repository CRAN

Date/Publication 2023-06-12 13:10:02 UTC

R topics documented:

`accident2014` *Sample of car accident location in the UK during year 2014.*

Description

Longitude and latitude of 500 car accident during year 2014 (source: www.data.gov.uk).

Usage

```
accident2014
```

Format

The dataset has 500 instances described by 2 variables (coordinates).

Source

https://www.data.gov.uk/

ADABOOST *Classification using AdaBoost*

Description

Ensemble learning, through AdaBoost Algorithm.

Usage

```
ADABOOST(
  x,
  y,
  learningmethod,
  nsamples = 100,
  fuzzy = FALSE,
  tune = FALSE,
  seed = NULL,
  ...
)
```

Arguments

x	The dataset (description/predictors), a matrix or data.frame.
y	The target (class labels or numeric values), a factor or vector.
learningmethod	The boosted method.
nsamples	The number of samplings.
fuzzy	Indicates whether or not fuzzy classification should be used or not.
tune	If true, the function returns paramters instead of a classification model.
seed	A specified seed for random number generation.
...	Other specific parameters for the leaning method.

Value

The classification model.

See Also

BAGGING, predict.boosting

Examples

```
## Not run:
require (datasets)
data (iris)
ADABOOST (iris [, -5], iris [, 5], NB)

## End(Not run)
```

alcohol *Alcohol dataset*

Description

This dataset has been extracted from the WHO database and depict the alcool habits in the 27 european contries (in 2010).

Usage

```
alcohol
```

Format

The dataset has 27 instances described by 4 variables. The variables are the average amount of alcool of different types per year par inhabitent.

Source

https://www.who.int/

APRIORI *Classification using APRIORI*

Description

This function builds a classification model using the association rules method APRIORI.

Usage

```
APRIORI(
  train,
  labels,
  supp = 0.05,
  conf = 0.8,
  prune = FALSE,
  tune = FALSE,
  ...
)
```

Arguments

train	The training set (description), as a data.frame.
labels	Class labels of the training set (vector or factor).
supp	The minimal support of an item set (numeric value).
conf	The minimal confidence of an item set (numeric value).
prune	A logical indicating whether to prune redundant rules or not (default: FALSE).
tune	If true, the function returns paramters instead of a classification model.
...	Other parameters.

Value

The classification model, as an object of class apriori.

See Also

predict.apriori, apriori-class, apriori

Examples

```
require ("datasets")
data (iris)
d = discretizeDF (iris,
   default = list (method = "interval", breaks = 3, labels = c ("small", "medium", "large")))
APRIORI (d [, -5], d [, 5], supp = .1, conf = .9, prune = TRUE)
```

---

apriori-class *APRIORI classification model*

Description

This class contains the classification model obtained by the APRIORI association rules method.

Slots

rules The set of rules obtained by APRIORI.

transactions The training set as a transaction object.

train The training set (description). A matrix or data.frame.

labels Class labels of the training set. Either a factor or an integer vector.

supp The minimal support of an item set (numeric value).

conf The minimal confidence of an item set (numeric value).

See Also

APRIORI, predict.apriori, print.apriori, summary.apriori, apriori

augmentation *Duplicate and add noise to a dataset*

Description

This function is a data augmentation technique. It duplicates rows and add gaussian noise to the duplicates.

Usage

```
augmentation(dataset, target, n = 5, sigma = 0.1, seed = NULL)
```

Arguments

dataset	The dataset to be split (data.frame or matrix).
target	The column index of the target variable (class label or response variable).
n	The scaling factor (as an integer value).
sigma	The baseline variance for the noise generation.
seed	A specified seed for random number generation.

Value

An augmented dataset.

Examples

```
require (datasets)
data (iris)
d = augmentation (iris, 5)
summary (iris)
summary (d)
```

autompg *Auto MPG dataset*

Description

This dataset was taken from the StatLib library which is maintained at Carnegie Mellon University. The dataset was used in the 1983 American Statistical Association Exposition.

Usage

```
autompg
```

Format

The dataset has 392 instances described by 8 variables. The seven first variables are numeric variables. The last variable is qualitative (car origin).

Source

https://archive.ics.uci.edu/ml/datasets/auto+mpg

BAGGING *Classification using Bagging*

Description

Ensemble learning, through Bagging Algorithm.

Usage

```
BAGGING(
  x,
  y,
  learningmethod,
  nsamples = 100,
  bag.size = nrow(x),
  seed = NULL,
  ...
)
```

Arguments

x	The dataset (description/predictors), a matrix or data.frame.
y	The target (class labels or numeric values), a factor or vector.
learningmethod	The boosted method.
nsamples	The number of samplings.
bag.size	The size of the samples.
seed	A specified seed for random number generation.
...	Other specific parameters for the leaning method.

Value

The classification model.

See Also

ADABOOST, predict.boosting

Examples

```
## Not run:
require (datasets)
data (iris)
BAGGING (iris [, -5], iris [, 5], NB)

## End(Not run)
```

`beetles` *Flea beetles dataset*

Description

Data were collected on the genus of flea beetle *Chaetocnema*, which contains three species: *concinna*, *heikertingeri*, and *heptapotamica*. Measurements were made on the width and angle of the aedeagus of each beetle. The goal of the original study was to form a classification rule to distinguish the three species.

Usage

```
beetles
```

Format

The dataset has 74 instances described by 3 variables. The variables are as follows:

`Width` The maximal width of aedeagus in the forpart (in microns).

`Angle` The front angle of the aedeagus (1 unit = 7.5 degrees).

`Shot.put` Species of flea beetle from the genus *Chaetocnema*.

Source

Lubischew, A.A. (1962) On the use of discriminant functions in taxonomy. Biometrics, 18, 455-477.

`birth` *Birth dataset*

Description

Tutorial data set (vector).

Usage

```
birth
```

Format

The dataset is a names vector of nine values (birth years).

boosting-class *Boosting methods model*

Description

This class contains the classification model obtained by the CDA method.

Slots

`models` List of models.

`x` The learning set.

`y` The target values.

See Also

ADABOOST, BAGGING, predict.boosting

boxclus *Clustering Box Plots*

Description

Produce a box-and-whisker plot for clustering results.

Usage

```
boxclus(d, clusters, legendpos = "topleft", ...)
```

Arguments

`d`	The dataset (`matrix` or `data.frame`).
`clusters`	Cluster labels of the training set (`vector` or `factor`).
`legendpos`	Position of the legend
`...`	Other parameters.

See Also

boxplot

Examples

```
require (datasets)
data (iris)
km = KMEANS (iris [, -5], k = 3)
boxclus (iris [, -5], km$cluster)
```

britpop *Population and location of 18 major british cities.*

Description

Longitude and latitude and population of 18 major cities in the Great Britain.

Usage

```
britpop
```

Format

The dataset has 18 instances described by 3 variables.

CA *Correspondence Analysis (CA)*

Description

Performs Correspondence Analysis (CA) including supplementary row and/or column points.

Usage

```
CA(
  d,
  ncp = 5,
  row.sup = NULL,
  col.sup = NULL,
  quanti.sup = NULL,
  quali.sup = NULL,
  row.w = NULL
)
```

Arguments

d	A ddata frame or a table with n rows and p columns, i.e. a contingency table.
ncp	The number of dimensions kept in the results (by default 5).
row.sup	A vector indicating the indexes of the supplementary rows.
col.sup	A vector indicating the indexes of the supplementary columns.
quanti.sup	A vector indicating the indexes of the supplementary continuous variables.
quali.sup	A vector indicating the indexes of the categorical supplementary variables.
row.w	An optional row weights (by default, a vector of 1 for uniform row weights); the weights are given only for the active individuals.

Value

The CA on the dataset.

See Also

CA, MCA, PCA, plot.factorial, factorial-class

Examples

```
data (children, package = "FactoMineR")
CA (children, row.sup = 15:18, col.sup = 6:8)
```

CART *Classification using CART*

Description

This function builds a classification model using CART.

Usage

```
CART(
  train,
  labels,
  minsplit = 1,
  maxdepth = log2(length(labels)),
  cp = NULL,
  tune = FALSE,
  ...
)
```

Arguments

train	The training set (description), as a data.frame.
labels	Class labels of the training set (vector or factor).
minsplit	The minimum leaf size during the learning.
maxdepth	Set the maximum depth of any node of the final tree, with the root node counted as depth 0.
cp	The complexity parameter of the tree. Cross-validation is used to determine optimal cp if NULL.
tune	If true, the function returns paramters instead of a classification model.
...	Other parameters.

Value

The classification model.

See Also

cartdepth, cartinfo, cartleafs, cartnodes, cartplot, rpart

Examples

```
require (datasets)
data (iris)
CART (iris [, -5], iris [, 5])
```

---

cartdepth *Depth*

Description

Return the dept of a decision tree.

Usage

```
cartdepth(model)
```

Arguments

model	The decision tree.

Value

The depth.

See Also

CART, cartinfo, cartleafs, cartnodes, cartplot

Examples

```
require (datasets)
data (iris)
model = CART (iris [, -5], iris [, 5])
cartdepth (model)
```

cartinfo *CART information*

Description

Return various information on a CART model.

Usage

```
cartinfo(model)
```

Arguments

model	The decision tree.

Value

Various information organized into a vector.

See Also

CART, cartdepth, cartleafs, cartnodes, cartplot

Examples

```
require (datasets)
data (iris)
model = CART (iris [, -5], iris [, 5])
cartinfo (model)
```

cartleafs *Number of Leafs*

Description

Return the number of leafs of a decision tree.

Usage

```
cartleafs(model)
```

Arguments

model	The decision tree.

Value

The number of leafs.

See Also

CART, cartdepth, cartinfo, cartnodes, cartplot

Examples

```
require (datasets)
data (iris)
model = CART (iris [, -5], iris [, 5])
cartleafs (model)
```

`cartnodes` *Number of Nodes*

Description

Return the number of nodes of a decision tree.

Usage

```
cartnodes(model)
```

Arguments

`model`	The decision tree.

Value

The number of nodes.

See Also

CART, cartdepth, cartinfo, cartleafs, cartplot

Examples

```
require (datasets)
data (iris)
model = CART (iris [, -5], iris [, 5])
cartnodes (model)
```

cartplot *CART Plot*

Description

Plot a decision tree obtained by CART.

Usage

```
cartplot(model, ...)
```

Arguments

model	The decision tree.
...	Other parameters.

See Also

CART, cartdepth, cartinfo, cartleafs, cartnodes

Examples

```
require (datasets)
data (iris)
model = CART (iris [, -5], iris [, 5])
cartplot (model)
```

CDA *Classification using Canonical Discriminant Analysis*

Description

This function builds a classification model using Canonical Discriminant Analysis.

Usage

```
CDA(train, labels, tune = FALSE, ...)
```

Arguments

train	The training set (description), as a data.frame.
labels	Class labels of the training set (vector or factor).
tune	If true, the function returns paramters instead of a classification model.
...	Other parameters.

Value

The classification model, as an object of class `glmnet`.

See Also

`plot.cda`, `predict.cda`, `cda-class`

Examples

```
require (datasets)
data (iris)
CDA (iris [, -5], iris [, 5])
```

`cda-class` *Canonical Disciminant Analysis model*

Description

This class contains the classification model obtained by the CDA method.

Slots

`proj` The projection of the dataset into the canonical base. A `data.frame`.

`transform` The transformation matrix between. A `matrix`.

`centers` Coordinates of the class centers. A `matrix`.

`within` The intr-class covarianc matrix. A `matrix`.

`eig` The eigen-values. A `matrix`.

`dim` The number of dimensions of the canonical base (numeric value).

`nb.classes` The number of clusters (numeric value).

`train` The training set (description). A `data.frame`.

`labels` Class labels of the training set. Either a `factor` or an integer `vector`.

`model` The prediction model.

See Also

`CDA`, `plot.cda`, `predict.cda`

closegraphics *Close a graphics device*

Description

Close the graphics device driver

Usage

```
closegraphics()
```

See Also

exportgraphics, toggleexport, dev.off

Examples

```
## Not run:
data (iris)
exportgraphics ("export.pdf")
plotdata (iris [, -5], iris [, 5])
closegraphics()

## End(Not run)
```

compare *Comparison of two sets of clusters*

Description

Comparison of two sets of clusters

Usage

```
compare(clus, gt, eval = "accuracy", comp = c("max", "pairwise", "cluster"))
```

Arguments

clus	The extracted clusters.
gt	The real clusters.
eval	The evluation criterion.
comp	Indicates whether a "max" or a "pairwise" evaluation should be used, or the evaluation for each individual "cluster".

Value

A numeric value indicating how much the two sets of clusters are similar.

See Also

compare.accuracy, compare.jaccard, compare.kappa, intern, stability

Examples

```
require (datasets)
data (iris)
km = KMEANS (iris [, -5], k = 3)
compare (km$cluster, iris [, 5])
## Not run:
compare (km$cluster, iris [, 5], eval = c ("accuracy", "kappa"), comp = "pairwise")

## End(Not run)
```

---

compare.accuracy *Comparison of two sets of clusters, using accuracy*

Description

Comparison of two sets of clusters, using accuracy

Usage

```
compare.accuracy(clus, gt, comp = c("max", "pairwise", "cluster"))
```

Arguments

clus	The extracted clusters.
gt	The real clusters.
comp	Indicates whether a "max" or a "pairwise" evaluation should be used, or the evaluation for each individual "cluster".

Value

A numeric value indicating how much the two sets of clusters are similar.

See Also

compare.jaccard, compare.kappa, compare

Examples

```
require (datasets)
data (iris)
km = KMEANS (iris [, -5], k = 3)
compare.accuracy (km$cluster, iris [, 5])
```

compare.jaccard *Comparison of two sets of clusters, using Jaccard index*

Description

Comparison of two sets of clusters, using Jaccard index

Usage

```
compare.jaccard(clus, gt, comp = c("max", "pairwise", "cluster"))
```

Arguments

clus	The extracted clusters.
gt	The real clusters.
comp	Indicates whether a "max" or a "pairwise" evaluation should be used, or the evaluation for each individual "cluster".

Value

A numeric value indicating how much the two sets of clusters are similar.

See Also

compare.accuracy, compare.kappa, compare

Examples

```
require (datasets)
data (iris)
km = KMEANS (iris [, -5], k = 3)
compare.jaccard (km$cluster, iris [, 5])
```

compare.kappa *Comparison of two sets of clusters, using kappa*

Description

Comparison of two sets of clusters, using kappa

Usage

```
compare.kappa(clus, gt, comp = c("max", "pairwise", "cluster"))
```

Arguments

clus	The extracted clusters.
gt	The real clusters.
comp	Indicates whether a "max" or a "pairwise" evaluation should be used, or the evaluation for each individual "cluster".

Value

A numeric value indicating how much the two sets of clusters are similar.

See Also

compare.accuracy, compare.jaccard, compare

Examples

```
require (datasets)
data (iris)
km = KMEANS (iris [, -5], k = 3)
compare.kappa (km$cluster, iris [, 5])
```

---

confusion *Confuion matrix*

Description

Plot a confusion matrix.

Usage

```
confusion(predictions, gt, norm = TRUE, graph = TRUE)
```

Arguments

predictions	The prediction.
gt	The ground truth.
norm	Whether or not the confusion matrix is normalized
graph	Whether or not a graphic is displayed.

Value

The confusion matrix.

See Also

evaluation, performance, splitdata

Examples

```
require ("datasets")
data (iris)
d = splitdata (iris, 5)
model = NB (d$train.x, d$train.y)
pred = predict (model, d$test.x)
confusion (d$test.y, pred)
```

---

cookies *Cookies dataset*

Description

This data set contains measurements from quantitative NIR spectroscopy. The example studied arises from an experiment done to test the feasibility of NIR spectroscopy to measure the composition of biscuit dough pieces (formed but unbaked biscuits). Two similar sample sets were made up, with the standard recipe varied to provide a large range for each of the four constituents under investigation: fat, sucrose, dry flour, and water. The calculated percentages of these four ingredients represent the 4 responses. There are 40 samples in the calibration or training set (with sample 23 being an outlier). There are a further 32 samples in the separate prediction or validation set (with example 21 considered as an outlier). An NIR reflectance spectrum is available for each dough piece. The spectral data consist of 700 points measured from 1100 to 2498 nanometers (nm) in steps of 2 nm.

Usage

```
cookies
cookies.desc.train
cookies.desc.test
cookies.y.train
cookies.y.test
```

Format

The cookies.desc.* datasets contains the 700 columns that correspond to the NIR reflectance spectrum. The cookies.y.* datasets contains four columns that correspond to the four constituents fat, sucrose, dry flour, and water. The cookies.*.train contains 40 rows that correspond to the calibration data. The cookies.*.test contains 32 rows that correspond to the prediction data.

Source

P. J. Brown and T. Fearn and M. Vannucci (2001) "Bayesian wavelet regression on curves with applications to a spectroscopic calibration problem", Journal of the American Statistical Association, 96(454), pp. 398-408.

See Also

labp, labc, nirp, nirc

cookplot *Plot the Cook's distance of a linear regression model*

Description

Plot the Cook's distance of a linear regression model.

Usage

```
cookplot(model, index = NULL, labels = NULL)
```

Arguments

model	The model to be plotted.
index	The index of the variable used for for the x-axis.
labels	The labels of the instances.

Examples

```
require (datasets)
data (trees)
model = LINREG (trees [, -3], trees [, 3])
cookplot (model)
```

correlated *Correlated variables*

Description

Return the list of correlated variables

Usage

```
correlated(d, threshold = 0.8)
```

Arguments

d	A data matrix.
threshold	The threshold on the (absolute) Pearson coefficient. If NULL, return the most correlated variables.

Value

The list of correlated variables (as a matrix of column names).

See Also

cor

Examples

```
data (iris)
correlated (iris)
```

`cost.curves` *Plot Cost Curves*

Description

This function plots Cost Curves of several classification predictions.

Usage

```
cost.curves(predictions, gt, methods.names = NULL)
```

Arguments

`predictions`	The predictions of a classification model (`factor` or `vector`).
`gt`	Actual labels of the dataset (`factor` or `vector`).
`methods.names`	The name of the compared methods (`vector`).

Value

The evaluation of the predictions (numeric value).

See Also

`roc.curves`, `performance`

Examples

```
require (datasets)
data (iris)
d = iris
levels (d [, 5]) = c ("+", "+", "-") # Building a two classes dataset
model.nb = NB (d [, -5], d [, 5])
model.lda = LDA (d [, -5], d [, 5])
pred.nb = predict (model.nb, d [, -5])
pred.lda = predict (model.lda, d [, -5])
cost.curves (cbind (pred.nb, pred.lda), d [, 5], c ("NB", "LDA"))
```

credit *Credit dataset*

Description

This is a fake dataset simulating a bank database about loan clients.

Usage

```
credit
```

Format

The dataset has 66 instances described by 11 qualitative variables.

data.diag *Square dataset*

Description

Generate a random dataset shaped like a square divided by a custom function

Usage

```
data.diag(
  n = 200,
  min = 0,
  max = 1,
  f = function(x) x,
  levels = NULL,
  graph = TRUE,
  seed = NULL
)
```

Arguments

n	Number of observations in the dataset.
min	Minimum value on each variables.
max	Maximum value on each variables.
f	The fucntion that separate the classes.
levels	Name of each class.
graph	A logical indicating whether or not a graphic should be plotted.
seed	A specified seed for random number generation.

Value

A randomly generated dataset.

See Also

data.parabol, data.target1, data.target2, data.twomoons, data.xor

Examples

```
data.diag ()
```

---

data.gauss *Gaussian mixture dataset*

Description

Generate a random multidimentional gaussian mixture.

Usage

```
data.gauss(
  n = 1000,
  k = 2,
  prob = rep(1/k, k),
  mu = cbind(rep(0, k), seq(from = 0, by = 3, length.out = k)),
  cov = rep(list(matrix(c(6, 0.9, 0.9, 0.3), ncol = 2, nrow = 2)), k),
  levels = NULL,
  graph = TRUE,
  seed = NULL
)
```

Arguments

n	Number of observations.
k	The number of classes.
prob	The a priori probability of each class.
mu	The means of the gaussian distributions.
cov	The covariance of the gaussian distributions.
levels	Name of each class.
graph	A logical indicating whether or not a graphic should be plotted.
seed	A specified seed for random number generation.

Value

A randomly generated dataset.

See Also

data.diag, data.parabol, data.target2, data.twomoons, data.xor

Examples

```
data.gauss ()
```

---

data.parabol *Parabol dataset*

Description

Generate a random dataset shaped like a parabol and a gaussian distribution

Usage

```
data.parabol(
  n = c(500, 100),
  xlim = c(-3, 3),
  center = c(0, 4),
  coeff = 0.5,
  sigma = c(0.5, 0.5),
  levels = NULL,
  graph = TRUE,
  seed = NULL
)
```

Arguments

n	Number of observations in each class.
xlim	Minimum and maximum on the x axis.
center	Coordinates of the center of the gaussian distribution.
coeff	Coefficient of the parabol.
sigma	Variance in each class.
levels	Name of each class.
graph	A logical indicating whether or not a graphic should be plotted.
seed	A specified seed for random number generation.

Value

A randomly generated dataset.

See Also

data.diag, data.target1, data.target2, data.twomoons, data.xor

Examples

```
data.parabol ()
```

`data.target1` *Target1 dataset*

Description

Generate a random dataset shaped like a target.

Usage

```
data.target1(
  r = 1:3,
  n = 200,
  sigma = 0.1,
  levels = NULL,
  graph = TRUE,
  seed = NULL
)
```

Arguments

`r`	Radius of each class.
`n`	Number of observations in each class.
`sigma`	Variance in each class.
`levels`	Name of each class.
`graph`	A logical indicating whether or not a graphic should be plotted.
`seed`	A specified seed for random number generation.

Value

A randomly generated dataset.

See Also

`data.diag`, `data.parabol`, `data.target2`, `data.twomoons`, `data.xor`

Examples

```
data.target1 ()
```

`data.target2` *Target2 dataset*

Description

Generate a random dataset shaped like a target.

Usage

```
data.target2(
  minr = c(0, 2),
  maxr = minr + 1,
  initn = 1000,
  levels = NULL,
  graph = TRUE,
  seed = NULL
)
```

Arguments

`minr`	Minimum radius of each class.
`maxr`	Maximum radius of each class.
`initn`	Number of observations at the beginning of the generation process.
`levels`	Name of each class.
`graph`	A logical indicating whether or not a graphic should be plotted.
`seed`	A specified seed for random number generation.

Value

A randomly generated dataset.

See Also

`data.diag`, `data.parabol`, `data.target1`, `data.twomoons`, `data.xor`

Examples

```
data.target2 ()
```

`data.twomoons` *Two moons dataset*

Description

Generate a random dataset shaped like two moons.

Usage

```
data.twomoons(
  r = 1,
  n = 200,
  sigma = 0.1,
  levels = NULL,
  graph = TRUE,
  seed = NULL
)
```

Arguments

`r`	Radius of each class.
`n`	Number of observations in each class.
`sigma`	Variance in each class.
`levels`	Name of each class.
`graph`	A logical indicating whether or not a graphic should be plotted.
`seed`	A specified seed for random number generation.

Value

A randomly generated dataset.

See Also

`data.diag`, `data.parabol`, `data.target1`, `data.target2`, `data.xor`

Examples

```
data.twomoons ()
```

`data.xor` *XOR dataset*

Description

Generate "XOR" dataset.

Usage

```
data.xor(
  n = 100,
  ndim = 2,
  sigma = 0.25,
  levels = NULL,
  graph = TRUE,
  seed = NULL
)
```

Arguments

`n`	Number of observations in each cluster.
`ndim`	The number of dimensions (2^ndim clusters are formed, grouped into two classes).
`sigma`	The variance.
`levels`	Name of each class.
`graph`	A logical indicating whether or not a graphic should be plotted.
`seed`	A specified seed for random number generation.

Value

A randomly generated dataset.

See Also

`data.diag`, `data.gauss`, `data.parabol`, `data.target2`, `data.twomoons`

Examples

```
data.xor ()
```

data1 *"data1" dataset*

Description

Synthetic dataset.

Usage

```
data1
```

Format

240 observations described by 4 variables and grouped into 16 classes.

Author(s)

Alexandre Blansché <alexandre.blansche@univ-lorraine.fr>

data2 *"data2" dataset*

Description

Synthetic dataset.

Usage

```
data2
```

Format

500 observations described by 10 variables and grouped into 3 classes.

Author(s)

Alexandre Blansché <alexandre.blansche@univ-lorraine.fr>

data3 *"data3" dataset*

Description

Synthetic dataset.

Usage

```
data3
```

Format

300 observations described by 3 variables and grouped into 3 classes.

Author(s)

Alexandre Blansché <alexandre.blansche@univ-lorraine.fr>

dataset-class *Training set and test set*

Description

This class contains a dataset divided into four parts: the training set and test set, description and class labels.

Slots

`train.x` the training set (description), as a `data.frame` or a `matrix`.

`train.y` the training set (target), as a `vector` or a `factor`.

`test.x` the training set (description), as a `data.frame` or a `matrix`.

`test.y` the training set (target), as a `vector` or a `factor`.

See Also

`splitdata`

dbs-class *DBSCAN model*

Description

This class contains the model obtained by the DBSCAN method.

Slots

`cluster` A vector of integers indicating the cluster to which each point is allocated.

`eps` Reachability distance (parameter).

`MinPts` Reachability minimum no. of points (parameter).

`isseed` A logical vector indicating whether a point is a seed (not border, not noise).

`data` The dataset that has been used to fit the map (as a `matrix`).

See Also

DBSCAN

DBSCAN *DBSCAN clustering method*

Description

Run the DBSCAN algorithm for clustering.

Usage

```
DBSCAN(d, minpts, epsilonDist, ...)
```

Arguments

`d`	The dataset (`matrix` or `data.frame`).
`minpts`	Reachability minimum no. of points.
`epsilonDist`	Reachability distance.
`...`	Other parameters.

Value

A clustering model obtained by DBSCAN.

See Also

dbscan, dbs-class, distplot, predict.dbs

Examples

```
require (datasets)
data (iris)
DBSCAN (iris [, -5], minpts = 5, epsilonDist = 1)
```

`decathlon` *Decathlon dataset*

Description

The dataset contains results from two athletics competitions. The 2004 Olympic Games in Athens and the 2004 Decastar.

Usage

```
decathlon
```

Format

The dataset has 41 instances described by 13 variables. The variables are as follows:

`100m` In seconds.

`Long.jump` In meters.

`Shot.put` In meters.

`High.jump` In meters.

`400m` In seconds.

`110m.h` In seconds.

`Discus.throw` In meters.

`Pole.vault` In meters.

`Javelin.throw` In meters.

`1500m` In seconds.

`Rank` The rank at the competition.

`Points` The number of points obtained by the athlete.

`Competition` `Olympics` or `Decastar`.

Source

https://husson.github.io/data.html

distplot *Plot a k-distance graphic*

Description

Plot the distance to the k's nearest neighbours of each object in decreasing order. Mostly used to determine the eps parameter for the dbscan function.

Usage

```
distplot(k, d, h = -1)
```

Arguments

k	The k parameter.
d	The dataset (matrix or data.frame).
h	The y-coordinate at which a horizontal line should be drawn.

See Also

DBSCAN, dbscan

Examples

```
require (datasets)
data (iris)
distplot (5, iris [, -5], h = .65)
```

EM *Expectation-Maximization clustering method*

Description

Run the EM algorithm for clustering.

Usage

```
EM(d, clusters, model = "VVV", ...)
```

Arguments

d	The dataset (matrix or data.frame).
clusters	Either an integer (the number of clusters) or a (vector) indicating the cluster to which each point is initially allocated.
model	A character string indicating the model. The help file for mclustModelNames describes the available models.
...	Other parameters.

Value

A clustering model obtained by EM.

See Also

em, mstep, mclustModelNames

Examples

```
require (datasets)
data (iris)
EM (iris [, -5], 3) # Default initialization
km = KMEANS (iris [, -5], k = 3)
EM (iris [, -5], km$cluster) # Initialization with another clustering method
```

---

em-class *Expectation-Maximization model*

Description

This class contains the model obtained by the EM method.

Slots

`modelName` A character string indicating the model. The help file for mclustModelNames describes the available models.

`prior` Specification of a conjugate prior on the means and variances.

`n` The number of observations in the dataset.

`d` The number of variables in the dataset.

`G` The number of components of the mixture.

`z` A matrix whose `[i,k]`th entry is the conditional probability of the ith observation belonging to the kth component of the mixture.

`parameters` A names list giving the parameters of the model.

`control` A list of control parameters for EM.

`loglik` The log likelihood for the data in the mixture model.

`cluster` A vector of integers (from `1:k`) indicating the cluster to which each point is allocated.

See Also

EM, mclustModelNames

eucalyptus *Eucalyptus dataset*

Description

Measuring the height of a tree is not an easy task. Is it possible to estimate the height as a function of the circumference of the trunk?

Usage

```
eucalyptus
```

Format

The dataset has 1429 instances (eucalyptus trees) with 2 measurements: the height and the circumference.

Source

http://www.cmap.polytechnique.fr/~lepennec/fr/teaching/

evaluation *Evaluation of classification or regression predictions*

Description

Evaluation predictions of a classification or a regression model.

Usage

```
evaluation(
  predictions,
  gt,
  eval = ifelse(is.factor(gt), "accuracy", "r2"),
  ...
)
```

Arguments

predictions	The predictions of a classification model (factor or vector).
gt	The ground truth of the dataset (factor or vector).
eval	The evaluation method.
...	Other parameters.

Value

The evaluation of the predictions (numeric value).

See Also

confusion, evaluation.accuracy, evaluation.fmeasure, evaluation.fowlkesmallows, evaluation.goodness, evaluation.jaccard, evaluation.kappa, evaluation.precision, evaluation.recall, evaluation.msep, evaluation.r2, performance

Examples

```
require (datasets)
data (iris)
d = splitdata (iris, 5)
model.nb = NB (d$train.x, d$train.y)
pred.nb = predict (model.nb, d$test.x)
# Default evaluation for classification
evaluation (pred.nb, d$test.y)
# Evaluation with two criteria
evaluation (pred.nb, d$test.y, eval = c ("accuracy", "kappa"))
data (trees)
d = splitdata (trees, 3)
model.linreg = LINREG (d$train.x, d$train.y)
pred.linreg = predict (model.linreg, d$test.x)
# Default evaluation for regression
evaluation (pred.linreg, d$test.y)
```

---

evaluation.accuracy *Accuracy of classification predictions*

Description

Evaluation predictions of a classification model according to accuracy.

Usage

```
evaluation.accuracy(predictions, gt, ...)
```

Arguments

predictions	The predictions of a classification model (factor or vector).
gt	The ground truth (factor or vector).
...	Other parameters.

Value

The evaluation of the predictions (numeric value).

See Also

evaluation.fmeasure, evaluation.fowlkesmallows, evaluation.goodness, evaluation.jaccard, evaluation.kappa, evaluation.precision, evaluation.precision, evaluation.recall, evaluation

Examples

```
require (datasets)
data (iris)
d = splitdata (iris, 5)
model.nb = NB (d$train.x, d$train.y)
pred.nb = predict (model.nb, d$test.x)
evaluation.accuracy (pred.nb, d$test.y)
```

---

evaluation.adjr2 *Adjusted R2 evaluation of regression predictions*

Description

Evaluation predictions of a regression model according to R2

Usage

```
evaluation.adjr2(predictions, gt, nrow = length(predictions), ncol, ...)
```

Arguments

predictions	The predictions of a regression model (vector).
gt	The ground truth (vector).
nrow	Number of observations.
ncol	Number of variables
...	Other parameters.

Value

The evaluation of the predictions (numeric value).

See Also

evaluation.msep, evaluation

Examples

```
require (datasets)
data (trees)
d = splitdata (trees, 3)
model.linreg = LINREG (d$train.x, d$train.y)
pred.linreg = predict (model.linreg, d$test.x)
evaluation.r2 (pred.linreg, d$test.y)
```

`evaluation.fmeasure` *F-measure*

Description

Evaluation predictions of a classification model according to the F-measure index.

Usage

```
evaluation.fmeasure(predictions, gt, beta = 1, positive = levels(gt)[1], ...)
```

Arguments

`predictions`	The predictions of a classification model (`factor` or `vector`).
`gt`	The ground truth (`factor` or `vector`).
`beta`	The weight given to precision.
`positive`	The label of the positive class.
`...`	Other parameters.

Value

The evaluation of the predictions (numeric value).

See Also

`evaluation.accuracy`, `evaluation.fowlkesmallows`, `evaluation.goodness`, `evaluation.jaccard`, `evaluation.kappa`, `evaluation.precision`, `evaluation.precision`, `evaluation.recall`, `evaluation`

Examples

```
require (datasets)
data (iris)
d = iris
levels (d [, 5]) = c ("+", "+", "-") # Building a two classes dataset
d = splitdata (d, 5)
model.nb = NB (d$train.x, d$train.y)
pred.nb = predict (model.nb, d$test.x)
evaluation.fmeasure (pred.nb, d$test.y)
```

`evaluation.fowlkesmallows` *Fowlkes–Mallows index*

Description

Evaluation predictions of a classification model according to the Fowlkes–Mallows index.

Usage

```
evaluation.fowlkesmallows(predictions, gt, positive = levels(gt)[1], ...)
```

Arguments

`predictions`	The predictions of a classification model (`factor` or `vector`).
`gt`	The ground truth (`factor` or `vector`).
`positive`	The label of the positive class.
`...`	Other parameters.

Value

The evaluation of the predictions (numeric value).

See Also

`evaluation.accuracy`, `evaluation.fmeasure`, `evaluation.goodness`, `evaluation.jaccard`, `evaluation.kappa`, `evaluation.precision`, `evaluation.precision`, `evaluation.recall`, `evaluation`

Examples

```
require (datasets)
data (iris)
d = iris
levels (d [, 5]) = c ("+", "+", "-") # Building a two classes dataset
d = splitdata (d, 5)
model.nb = NB (d$train.x, d$train.y)
pred.nb = predict (model.nb, d$test.x)
evaluation.fowlkesmallows (pred.nb, d$test.y)
```

`evaluation.goodness` *Goodness*

Description

Evaluation predictions of a classification model according to Goodness index.

Usage

```
evaluation.goodness(predictions, gt, beta = 1, positive = levels(gt)[1], ...)
```

Arguments

`predictions`	The predictions of a classification model (`factor` or `vector`).
`gt`	The ground truth (`factor` or `vector`).
`beta`	The weight given to precision.
`positive`	The label of the positive class.
`...`	Other parameters.

Value

The evaluation of the predictions (numeric value).

See Also

`evaluation.accuracy`, `evaluation.fmeasure`, `evaluation.fowlkesmallows`, `evaluation.jaccard`, `evaluation.kappa`, `evaluation.precision`, `evaluation.precision`, `evaluation.recall`, `evaluation`

Examples

```
require (datasets)
data (iris)
d = iris
levels (d [, 5]) = c ("+", "+", "-") # Building a two classes dataset
d = splitdata (d, 5)
model.nb = NB (d$train.x, d$train.y)
pred.nb = predict (model.nb, d$test.x)
evaluation.goodness (pred.nb, d$test.y)
```

`evaluation.jaccard` *Jaccard index*

Description

Evaluation predictions of a classification model according to Jaccard index.

Usage

```
evaluation.jaccard(predictions, gt, positive = levels(gt)[1], ...)
```

Arguments

`predictions`	The predictions of a classification model (`factor` or `vector`).
`gt`	The ground truth (`factor` or `vector`).
`positive`	The label of the positive class.
`...`	Other parameters.

Value

The evaluation of the predictions (numeric value).

See Also

`evaluation.accuracy`, `evaluation.fmeasure`, `evaluation.fowlkesmallows`, `evaluation.goodness`, `evaluation.kappa`, `evaluation.precision`, `evaluation.precision`, `evaluation.recall`, `evaluation`

Examples

```
require (datasets)
data (iris)
d = iris
levels (d [, 5]) = c ("+", "+", "-") # Building a two classes dataset
d = splitdata (d, 5)
model.nb = NB (d$train.x, d$train.y)
pred.nb = predict (model.nb, d$test.x)
evaluation.jaccard (pred.nb, d$test.y)
```

evaluation.kappa *Kappa evaluation of classification predictions*

Description

Evaluation predictions of a classification model according to kappa.

Usage

```
evaluation.kappa(predictions, gt, ...)
```

Arguments

predictions	The predictions of a classification model (factor or vector).
gt	The ground truth (factor or vector).
...	Other parameters.

Value

The evaluation of the predictions (numeric value).

See Also

evaluation.accuracy, evaluation.fmeasure, evaluation.fowlkesmallows, evaluation.goodness, evaluation.jaccard, evaluation.kappa, evaluation.precision, evaluation.precision, evaluation.recall, evaluation

Examples

```
require (datasets)
data (iris)
d = splitdata (iris, 5)
model.nb = NB (d$train.x, d$train.y)
pred.nb = predict (model.nb, d$test.x)
evaluation.kappa (pred.nb, d$test.y)
```

evaluation.msep *MSEP evaluation of regression predictions*

Description

Evaluation predictions of a regression model according to MSEP

Usage

```
evaluation.msep(predictions, gt, ...)
```

Arguments

predictions	The predictions of a regression model (vector).
gt	The ground truth (vector).
...	Other parameters.

Value

The evaluation of the predictions (numeric value).

See Also

evaluation.r2, evaluation

Examples

```
require (datasets)
data (trees)
d = splitdata (trees, 3)
model.lin = LINREG (d$train.x, d$train.y)
pred.lin = predict (model.lin, d$test.x)
evaluation.msep (pred.lin, d$test.y)
```

---

evaluation.precision *Precision of classification predictions*

Description

Evaluation predictions of a classification model according to precision. Works only for two classes problems.

Usage

```
evaluation.precision(predictions, gt, positive = levels(gt)[1], ...)
```

Arguments

predictions	The predictions of a classification model (factor or vector).
gt	The ground truth (factor or vector).
positive	The label of the positive class.
...	Other parameters.

Value

The evaluation of the predictions (numeric value).

See Also

evaluation.accuracy, evaluation.fmeasure, evaluation.fowlkesmallows, evaluation.goodness, evaluation.jaccard, evaluation.kappa, evaluation.recall,evaluation

Examples

```
require (datasets)
data (iris)
d = iris
levels (d [, 5]) = c ("+", "+", "-") # Building a two classes dataset
d = splitdata (d, 5)
model.nb = NB (d$train.x, d$train.y)
pred.nb = predict (model.nb, d$test.x)
evaluation.precision (pred.nb, d$test.y)
```

---

evaluation.r2 *R2 evaluation of regression predictions*

Description

Evaluation predictions of a regression model according to R2

Usage

```
evaluation.r2(predictions, gt, ...)
```

Arguments

predictions	The predictions of a regression model (vector).
gt	The ground truth (vector).
...	Other parameters.

Value

The evaluation of the predictions (numeric value).

See Also

evaluation.msep, evaluation

Examples

```
require (datasets)
data (trees)
d = splitdata (trees, 3)
model.linreg = LINREG (d$train.x, d$train.y)
pred.linreg = predict (model.linreg, d$test.x)
evaluation.r2 (pred.linreg, d$test.y)
```

`evaluation.recall` *Recall of classification predictions*

Description

Evaluation predictions of a classification model according to recall. Works only for two classes problems.

Usage

```
evaluation.recall(predictions, gt, positive = levels(gt)[1], ...)
```

Arguments

`predictions`	The predictions of a classification model (`factor` or `vector`).
`gt`	The ground truth (`factor` or `vector`).
`positive`	The label of the positive class.
`...`	Other parameters.

Value

The evaluation of the predictions (numeric value).

See Also

`evaluation.accuracy`, `evaluation.fmeasure`, `evaluation.fowlkesmallows`, `evaluation.goodness`, `evaluation.jaccard`, `evaluation.kappa`, `evaluation.precision`, `evaluation`

Examples

```
require (datasets)
data (iris)
d = iris
levels (d [, 5]) = c ("+", "+", "-") # Building a two classes dataset
d = splitdata (d, 5)
model.nb = NB (d$train.x, d$train.y)
pred.nb = predict (model.nb, d$test.x)
evaluation.recall (pred.nb, d$test.y)
```

exportgraphics *Open a graphics device*

Description

Starts the graphics device driver

Usage

```
exportgraphics(file, type = tail(strsplit(file, split = "\\.")[[1]], 1), ...)
```

Arguments

file	A character string giving the name of the file.
type	The type of graphics device.
...	Other parameters.

See Also

closegraphics, toggleexport, Devices

Examples

```
## Not run:
data (iris)
exportgraphics ("export.pdf")
plotdata (iris [, -5], iris [, 5])
closegraphics()

## End(Not run)
```

exportgraphics.off *Toggle graphic exports*

Description

Toggle graphic exports on and off

Usage

```
exportgraphics.off()

exportgraphics.on()

toggleexport(export = NULL)

toggleexport.off()

toggleexport.on()
```

Arguments

export	If TRUE, exports are activated, if FALSE, exports are deactivated. If null, switches on and off.

See Also

closegraphics, exportgraphics

Examples

```
## Not run:
data (iris)
toggleexport (FALSE)
exportgraphics ("export.pdf")
plotdata (iris [, -5], iris [, 5])
closegraphics()
toggleexport (TRUE)
exportgraphics ("export.pdf")
plotdata (iris [, -5], iris [, 5])
closegraphics()

## End(Not run)
```

---

factorial-class *Factorial analysis results*

Description

This class contains the classification model obtained by the CDA method.

See Also

CA, MCA, PCA, plot.factorial

FEATURESELECTION *Classification with Feature selection*

Description

Apply a classification method after a subset of features has been selected.

Usage

```
FEATURESELECTION(
  train,
  labels,
  algorithm = c("ranking", "forward", "backward", "exhaustive"),
 unieval = if (algorithm[1] == "ranking") c("fisher", "fstat", "relief", "inertiaratio")
    else NULL,
  uninb = NULL,
  unithreshold = NULL,
 multieval = if (algorithm[1] == "ranking") NULL else c("cfs", "fstat", "inertiaratio",
    "wrapper"),
  wrapmethod = NULL,
  mainmethod = wrapmethod,
  tune = FALSE,
  ...
)
```

Arguments

`train`	The training set (description), as a `data.frame`.
`labels`	Class labels of the training set (`vector` or `factor`).
`algorithm`	The feature selection algorithm.
`unieval`	The (univariate) evaluation criterion. `uninb`, `unithreshold` or `multieval` must be specified.
`uninb`	The number of selected feature (univariate evaluation).
`unithreshold`	The threshold for selecting feature (univariate evaluation).
`multieval`	The (multivariate) evaluation criterion.
`wrapmethod`	The classification method used for the wrapper evaluation.
`mainmethod`	The final method used for data classification. If a wrapper evaluation is used, the same classification method should be used.
`tune`	If true, the function returns paramters instead of a classification model.
`...`	Other parameters.

See Also

`selectfeatures`, `predict.selection`, `selection-class`

Examples

```
## Not run:
require (datasets)
data (iris)
FEATURESELECTION (iris [, -5], iris [, 5], uninb = 2, mainmethod = LDA)

## End(Not run)
```

`filter.rules` *Filtering a set of rules*

Description

This function facilitate the selection of a subset from a set of rules.

Usage

```
filter.rules(
  rules,
  pattern = NULL,
  left = pattern,
  right = pattern,
  removeMatches = FALSE
)
```

Arguments

`rules`	A set of rules.
`pattern`	A pattern to match (antecedent and consequent): a character string.
`left`	A pattern to match (antecedent only): a character string.
`right`	A pattern to match (consequent only): a character string.
`removeMatches`	A logical indicating whether to remove matching rules (`TRUE`) or to keep those (`FALSE`).

Value

The filtered set of rules.

See Also

`apriori`, `subset`

Examples

```
require ("arules")
data ("Adult")
r = apriori (Adult)
filter.rules (r, right = "marital-status=")
subset (r, subset = rhs %pin% "marital-status=")
```

`frequentwords` *Frequent words*

Description

Most frequent words of the corpus.

Usage

```
frequentwords(
  corpus,
  nb,
  mincount = 5,
  minphrasecount = NULL,
  ngram = 1,
  lang = "en",
  stopwords = lang
)
```

Arguments

`corpus`	The corpus of documents (a vector of characters) or the vocabulary of the documents (result of function `getvocab`).
`nb`	The number of words to be returned.
`mincount`	Minimum word count to be considered as frequent.
`minphrasecount`	Minimum collocation of words count to be considered as frequent.
`ngram`	maximum size of n-grams.
`lang`	The language of the documents (NULL if no stemming).
`stopwords`	Stopwords, or the language of the documents. NULL if stop words should not be removed.

Value

The most frequent words of the corpus.

See Also

`getvocab`

Examples

```
## Not run:
text = loadtext ("http://mattmahoney.net/dc/text8.zip")
frequentwords (text, 100)
vocab = getvocab (text)
frequentwords (vocab, 100)

## End(Not run)
```

`general.rules` *Remove redundancy in a set of rules*

Description

This function remove every redundant rules, keeping only the most general ones.

Usage

```
general.rules(r)
```

Arguments

`r`	A set of rules.

Value

A set of rules, without redundancy.

See Also

apriori

Examples

```
require ("arules")
data ("Adult")
r = apriori (Adult)
inspect (general.rules (r))
```

getvocab *Extract words and phrases from a corpus*

Description

Extract words and phrases from a corpus of documents.

Usage

```
getvocab(
  corpus,
  mincount = 5,
  minphrasecount = NULL,
  ngram = 1,
  lang = "en",
  stopwords = lang,
  ...
)
```

Arguments

corpus	The corpus of documents (a vector of characters).
mincount	Minimum word count to be considered as frequent.
minphrasecount	Minimum collocation of words count to be considered as frequent.
ngram	maximum size of n-grams.
lang	The language of the documents (NULL if no stemming).
stopwords	Stopwords, or the language of the documents. NULL if stop words should not be removed.
...	Other parameters.

Value

The vocabulary used in the corpus of documents.

See Also

plotzipf, stopwords, create_vocabulary

Examples

```
## Not run:
text = loadtext ("http://mattmahoney.net/dc/text8.zip")
vocab1 = getvocab (text) # With stemming
nrow (vocab1)
vocab2 = getvocab (text, lang = NULL) # Without stemming
nrow (vocab2)

## End(Not run)
```

GRADIENTBOOSTING *Classification using Gradient Boosting*

Description

This function builds a classification model using Gradient Boosting

Usage

```
GRADIENTBOOSTING(
  train,
  labels,
  ntree = 500,
  learningrate = 0.3,
  tune = FALSE,
  ...
)
```

Arguments

train	The training set (description), as a data.frame.
labels	Class labels of the training set (vector or factor).
ntree	The number of trees in the forest.
learningrate	The learning rate (between 0 and 1).
tune	If true, the function returns paramters instead of a classification model.
...	Other parameters.

Value

The classification model.

See Also

xgboost

Examples

```
## Not run:
require (datasets)
data (iris)
GRADIENTBOOSTING (iris [, -5], iris [, 5])

## End(Not run)
```

HCA *Hierarchical Cluster Analysis method*

Description

Run the HCA method for clustering.

Usage

```
HCA(d, method = c("ward", "single"), k = NULL, ...)
```

Arguments

d	The dataset (matrix or data.frame).
method	Character string defining the clustering method.
k	The number of cluster.
...	Other parameters.

Value

The cluster hierarchy (hca object).

See Also

agnes

Examples

```
require (datasets)
data (iris)
HCA (iris [, -5], method = "ward", k = 3)
```

intern *Clustering evaluation through internal criteria*

Description

Evaluation a clustering algorithm according to internal criteria.

Usage

```
intern(clus, d, eval = "intraclass", type = c("global", "cluster"))
```

Arguments

clus	The extracted clusters.
d	The dataset.
eval	The evaluation criteria.
type	Indicates whether a "global" or a "cluster"-wise evaluation should be used.

Value

The evaluation of the clustering.

See Also

compare, stability, intern.dunn, intern.interclass, intern.intraclass

Examples

```
require (datasets)
data (iris)
km = KMEANS (iris [, -5], k = 3)
intern (km$clus, iris [, -5])
intern (km$clus, iris [, -5], type = "cluster")
intern (km$clus, iris [, -5], eval = c ("intraclass", "interclass"))
intern (km$clus, iris [, -5], eval = c ("intraclass", "interclass"), type = "cluster")
```

---

intern.dunn *Clustering evaluation through Dunn's index*

Description

Evaluation a clustering algorithm according to Dunn's index.

Usage

```
intern.dunn(clus, d, type = c("global"))
```

Arguments

clus	The extracted clusters.
d	The dataset.
type	Indicates whether a "global" or a "cluster"-wise evaluation should be used.

Value

The evaluation of the clustering.

See Also

intern, intern.interclass, intern.intraclass

Examples

```
require (datasets)
data (iris)
km = KMEANS (iris [, -5], k = 3)
intern.dunn (km$clus, iris [, -5])
```

`intern.interclass` *Clustering evaluation through interclass inertia*

Description

Evaluation a clustering algorithm according to interclass inertia.

Usage

```
intern.interclass(clus, d, type = c("global", "cluster"))
```

Arguments

`clus`	The extracted clusters.
`d`	The dataset.
`type`	Indicates whether a "global" or a "cluster"-wise evaluation should be used.

Value

The evaluation of the clustering.

See Also

intern, intern.dunn, intern.intraclass

Examples

```
require (datasets)
data (iris)
km = KMEANS (iris [, -5], k = 3)
intern.interclass (km$clus, iris [, -5])
```

intern.intraclass *Clustering evaluation through intraclass inertia*

Description

Evaluation a clustering algorithm according to intraclass inertia.

Usage

```
intern.intraclass(clus, d, type = c("global", "cluster"))
```

Arguments

clus	The extracted clusters.
d	The dataset.
type	Indicates whether a "global" or a "cluster"-wise evaluation should be used.

Value

The evaluation of the clustering.

See Also

intern, intern.dunn, intern.interclass

Examples

```
require (datasets)
data (iris)
km = KMEANS (iris [, -5], k = 3)
intern.intraclass (km$clus, iris [, -5])
```

ionosphere *Ionosphere dataset*

Description

This is a dataset from the UCI repository. This radar data was collected by a system in Goose Bay, Labrador. This system consists of a phased array of 16 high-frequency antennas with a total transmitted power on the order of 6.4 kilowatts. See the paper for more details. The targets were free electrons in the ionosphere. "Good" radar returns are those showing evidence of some type of structure in the ionosphere. "Bad" returns are those that do not; their signals pass through the ionosphere. Received signals were processed using an autocorrelation function whose arguments are the time of a pulse and the pulse number. There were 17 pulse numbers for the Goose Bay system. Instances in this databse are described by 2 attributes per pulse number, corresponding to the complex values returned by the function resulting from the complex electromagnetic signal. One attribute with constant value has been removed.

Usage

```
ionosphere
```

Format

The dataset has 351 instances described by 34. The last variable is the class.

Source

https://archive.ics.uci.edu/ml/datasets/ionosphere

---

kaiser *Kaiser rule*

Description

Apply the Kaiser rule to determine the appropriate number of PCA axes.

Usage

```
kaiser(pca)
```

Arguments

pca	The PCA result (object of class `factorial-class`).

See Also

PCA, factorial-class

Examples

```
require (datasets)
data (iris)
pca = PCA (iris, quali.sup = 5)
kaiser (pca)
```

KERREG *Kernel Regression*

Description

This function builds a kernel regression model.

Usage

```
KERREG(x, y, bandwidth = 1, tune = FALSE, ...)
```

Arguments

x	Predictor matrix.
y	Response vector.
bandwidth	The bandwidth parameter.
tune	If true, the function returns paramters instead of a classification model.
...	Other parameters.

Value

The classification model, as an object of class model-class.

See Also

npregress

Examples

```
require (datasets)
data (trees)
KERREG (trees [, -3], trees [, 3])
```

KMEANS *K-means method*

Description

Run K-means for clustering.

Usage

```
KMEANS(
  d,
  k = 9,
  criterion = c("none", "pseudo-F"),
  graph = FALSE,
  nstart = 10,
  ...
)
```

Arguments

d	The dataset (matrix or data.frame).
k	The number of cluster.
criterion	The criterion for cluster number selection. If none, k is used, if not the number of cluster is selected between 2 and k.
graph	A logical indicating whether or not a graphic should be plotted (cluster number selection).
nstart	Define how many random sets should be chosen.
...	Other parameters.

Value

The clustering (kmeans object).

See Also

kmeans, predict.kmeans

Examples

```
require (datasets)
data (iris)
KMEANS (iris [, -5], k = 3)
KMEANS (iris [, -5], criterion = "pseudo-F") # With automatic detection of the nmber of clusters
```

---

kmeans.getk *Estimation of the number of clusters for* K*-means*

Description

Estimate the optimal number of cluster of the *K*-means clustering method.

Usage

```
kmeans.getk(
  d,
  max = 9,
  criterion = "pseudo-F",
  graph = TRUE,
  nstart = 10,
  seed = NULL
)
```

Arguments

d	The dataset (matrix or data.frame).
max	The maximum number of clusters. Values from 2 to max are evaluated.
criterion	The criterion to be optimized. "pseudo-F" is the only criterion implemented in the current version.
graph	A logical indicating whether or not a graphic should be plotted.
nstart	The number of random sets chosen for kmeans initialization.
seed	A specified seed for random number generation.

Value

The optimal number of cluster of the *K*-means clustering method according to the chosen criterion.

See Also

pseudoF, kmeans

Examples

```
require (datasets)
data (iris)
kmeans.getk (iris [, -5])
```

KNN *Classification using k-NN*

Description

This function builds a classification model using Logistic Regression.

Usage

```
KNN(train, labels, k = 1:10, tune = FALSE, ...)
```

Arguments

train	The training set (description), as a data.frame.
labels	Class labels of the training set (vector or factor).
k	The k parameter.
tune	If true, the function returns paramters instead of a classification model.
...	Other parameters.

Value

The classification model.

See Also

knn

Examples

```
require (datasets)
data (iris)
KNN (iris [, -5], iris [, 5])
```

---

knn-class *K Nearest Neighbours model*

Description

This class contains the classification model obtained by the k-NN method.

Slots

train The training set (description). A data.frame.

labels Class labels of the training set. Either a factor or an integer vector.

k The k parameter.

See Also

KNN, predict.knn

LDA *Classification using Linear Discriminant Analysis*

Description

This function builds a classification model using Linear Discriminant Analysis.

Usage

```
LDA(train, labels, tune = FALSE, ...)
```

Arguments

train	The training set (description), as a data.frame.
labels	Class labels of the training set (vector or factor).
tune	If true, the function returns paramters instead of a classification model.
...	Other parameters.

Value

The classification model.

See Also

lda

Examples

```
require (datasets)
data (iris)
LDA (iris [, -5], iris [, 5])
```

leverageplot *Plot the leverage points of a linear regression model*

Description

Plot the leverage points of a linear regression model.

Usage

```
leverageplot(model, index = NULL, labels = NULL)
```

Arguments

model	The model to be plotted.
index	The index of the variable used for for the x-axis.
labels	The labels of the instances.

Examples

```
require (datasets)
data (trees)
model = LINREG (trees [, -3], trees [, 3])
leverageplot (model)
```

LINREG *Linear Regression*

Description

This function builds a linear regression model. Standard least square method, variable selection, factorial methods are available.

Usage

```
LINREG(
  x,
  y,
  quali = c("none", "intercept", "slope", "both"),
  reg = c("linear", "subset", "ridge", "lasso", "elastic", "pcr", "plsr"),
  regeval = c("r2", "bic", "adjr2", "cp", "msep"),
  scale = TRUE,
  lambda = 10^seq(-5, 5, length.out = 101),
  alpha = 0.5,
  graph = TRUE,
  tune = FALSE,
  ...
)
```

Arguments

x	Predictor matrix.
y	Response vector.
quali	Indicates how to use the qualitative variables.
reg	The algorithm.
regeval	The evaluation criterion for subset selection.
scale	If true, PCR and PLS use scaled dataset.
lambda	The lambda parameter of Ridge, Lasso and Elastic net regression.

alpha	The elasticnet mixing parameter.
graph	A logical indicating whether or not graphics should be plotted (ridge, LASSO and elastic net).
tune	If true, the function returns paramters instead of a classification model.
...	Other parameters.

Value

The classification model, as an object of class model-class.

See Also

lm, regsubsets, mvr, glmnet

Examples

```
## Not run:
require (datasets)
# With one independant variable
data (cars)
LINREG (cars [, -2], cars [, 2])
# With two independant variables
data (trees)
LINREG (trees [, -3], trees [, 3])
# With non numeric variables
data (ToothGrowth)
LINREG (ToothGrowth [, -1], ToothGrowth [, 1], quali = "intercept") # Different intersept
LINREG (ToothGrowth [, -1], ToothGrowth [, 1], quali = "slope") # Different slope
LINREG (ToothGrowth [, -1], ToothGrowth [, 1], quali = "both") # Complete model
# With multiple numeric variables
data (mtcars)
LINREG (mtcars [, -1], mtcars [, 1])
LINREG (mtcars [, -1], mtcars [, 1], reg = "subset", regeval = "adjr2")
LINREG (mtcars [, -1], mtcars [, 1], reg = "ridge")
LINREG (mtcars [, -1], mtcars [, 1], reg = "lasso")
LINREG (mtcars [, -1], mtcars [, 1], reg = "elastic")
LINREG (mtcars [, -1], mtcars [, 1], reg = "pcr")
LINREG (mtcars [, -1], mtcars [, 1], reg = "plsr")

## End(Not run)
```

---

linsep *Linsep dataset*

Description

Synthetic dataset.

Usage

```
linsep
```

Format

Class A contains 50 observations and class B contains 500 observations. There are two numeric variables: X and Y.

Author(s)

Alexandre Blansché <alexandre.blansche@univ-lorraine.fr>

---

loadtext *load a text file*

Description

(Down)Load a text file (and extract it if it is in a zip file).

Usage

```
loadtext(
  file = file.choose(),
  dir = "~/",
  collapse = TRUE,
  sep = NULL,
  categories = NULL
)
```

Arguments

file	The path or URL of the text file.
dir	The (temporary) directory, where the file is downloaded. The file is deleted at the end of this function.
collapse	Indicates whether or not lines of each documents should collapse together or not.
sep	Separator between text fields.
categories	Columns that should be considered as categorial data.

Value

The text contained in the dowloaded file.

See Also

download.file, unzip

Examples

```
## Not run:
text = loadtext ("http://mattmahoney.net/dc/text8.zip")

## End(Not run)
```

LR *Classification using Logistic Regression*

Description

This function builds a classification model using Logistic Regression.

Usage

```
LR(train, labels, tune = FALSE, ...)
```

Arguments

`train`	The training set (description), as a `data.frame`.
`labels`	Class labels of the training set (`vector` or `factor`).
`tune`	If true, the function returns paramters instead of a classification model.
`...`	Other parameters.

Value

The classification model.

See Also

`multinom`

Examples

```
require (datasets)
data (iris)
LR (iris [, -5], iris [, 5])
```

MCA *Multiple Correspondence Analysis (MCA)*

Description

Performs Multiple Correspondence Analysis (MCA) with supplementary individuals, supplementary quantitative variables and supplementary categorical variables. Performs also Specific Multiple Correspondence Analysis with supplementary categories and supplementary categorical variables. Missing values are treated as an additional level, categories which are rare can be ventilated.

Usage

```
MCA(
  d,
  ncp = 5,
  ind.sup = NULL,
  quanti.sup = NULL,
  quali.sup = NULL,
  row.w = NULL
)
```

Arguments

d	A ddata frame or a table with n rows and p columns, i.e. a contingency table.
ncp	The number of dimensions kept in the results (by default 5).
ind.sup	A vector indicating the indexes of the supplementary individuals.
quanti.sup	A vector indicating the indexes of the quantitative supplementary variables.
quali.sup	A vector indicating the indexes of the categorical supplementary variables.
row.w	An optional row weights (by default, a vector of 1 for uniform row weights); the weights are given only for the active individuals.

Value

The MCA on the dataset.

See Also

MCA, CA, PCA, plot.factorial, factorial-class

Examples

```
data (tea, package = "FactoMineR")
MCA (tea, quanti.sup = 19, quali.sup = 20:36)
```

MEANSHIFT *MeanShift method*

Description

Run MeanShift for clustering.

Usage

```
MEANSHIFT(
  d,
  mskernel = "NORMAL",
  bandwidth = rep(1, ncol(d)),
  alpha = 0,
  iterations = 10,
  epsilon = 1e-08,
  epsilonCluster = 1e-04,
  ...
)
```

Arguments

d	The dataset (matrix or data.frame).
mskernel	A string indicating the kernel associated with the kernel density estimate that the mean shift is optimizing over.
bandwidth	Used in the kernel density estimate for steepest ascent classification.
alpha	A scalar tuning parameter for normal kernels.
iterations	The number of iterations to perform mean shift.
epsilon	A scalar used to determine when to terminate the iteration of a individual query point.
epsilonCluster	A scalar used to determine the minimum distance between distinct clusters.
...	Other parameters.

Value

The clustering (meanshift object).

See Also

meanShift, predict.meanshift

Examples

```
## Not run:
require (datasets)
data (iris)
MEANSHIFT (iris [, -5], bandwidth = .75)

## End(Not run)
```

`meanshift-class` *MeanShift model*

Description

This class contains the model obtained by the MEANSHIFT method.

Slots

`cluster` A vector of integers indicating the cluster to which each point is allocated.

`value` A vector or matrix containing the location of the classified local maxima in the support.

`data` The leaning set.

`kernel` A string indicating the kernel associated with the kernel density estimate that the mean shift is optimizing over.

`bandwidth` Used in the kernel density estimate for steepest ascent classification.

`alpha` A scalar tuning parameter for normal kernels.

`iterations` The number of iterations to perform mean shift.

`epsilon` A scalar used to determine when to terminate the iteration of a individual query point.

`epsilonCluster` A scalar used to determine the minimum distance between distinct clusters.

See Also

MEANSHIFT

`MLP` *Classification using Multilayer Perceptron*

Description

This function builds a classification model using Multilayer Perceptron.

Usage

```
MLP(
  train,
  labels,
  hidden = ifelse(is.vector(train), 2:(1 + nlevels(labels)), 2:(ncol(train) +
    nlevels(labels))),
  decay = 10^(-3:-1),
  methodparameters = NULL,
  tune = FALSE,
  ...
)
```

Arguments

train	The training set (description), as a data.frame.
labels	Class labels of the training set (vector or factor).
hidden	The size of the hidden layer (if a vector, cross-over validation is used to chose the best size).
decay	The decay (between 0 and 1) of the backpropagation algorithm (if a vector, cross-over validation is used to chose the best size).
methodparameters	Object containing the parameters. If given, it replaces size and decay.
tune	If true, the function returns paramters instead of a classification model.
...	Other parameters.

Value

The classification model.

See Also

nnet

Examples

```
## Not run:
require (datasets)
data (iris)
MLP (iris [, -5], iris [, 5], hidden = 4, decay = .1)

## End(Not run)
```

MLPREG *Multi-Layer Perceptron Regression*

Description

This function builds a regression model using MLP.

Usage

```
MLPREG(
  x,
  y,
  size = 2:(ifelse(is.vector(x), 2, ncol(x))),
  decay = 10^(-3:-1),
  params = NULL,
  tune = FALSE,
  ...
)
```

Arguments

x	Predictor matrix.
y	Response vector.
size	The size of the hidden layer (if a vector, cross-over validation is used to chose the best size).
decay	The decay (between 0 and 1) of the backpropagation algorithm (if a vector, cross-over validation is used to chose the best size).
params	Object containing the parameters. If given, it replaces size and decay.
tune	If true, the function returns paramters instead of a classification model.
...	Other parameters.

Value

The classification model, as an object of class model-class.

See Also

nnet

Examples

```
## Not run:
require (datasets)
data (trees)
MLPREG (trees [, -3], trees [, 3])

## End(Not run)
```

model-class *Generic classification or regression model*

Description

This is a wrapper class containing the classification model obtained by any classification or regression method.

Slots

`model` The wrapped model.

`method` The name of the method.

See Also

`predict.model`, `predict`

movies *Movies dataset*

Description

Extract from the movie lens dataset. Missing values have been imputed.

Usage

```
movies
```

Format

A set of 49 movies, rated by 55 users.

Source

https://grouplens.org/datasets/movielens/

NB *Classification using Naive Bayes*

Description

This function builds a classification model using Naive Bayes.

Usage

```
NB(train, labels, tune = FALSE, ...)
```

Arguments

train	The training set (description), as a data.frame.
labels	Class labels of the training set (vector or factor).
tune	If true, the function returns paramters instead of a classification model.
...	Other parameters.

Value

The classification model.

See Also

naiveBayes

Examples

```
require (datasets)
data (iris)
NB (iris [, -5], iris [, 5])
```

NMF *Non-negative Matrix Factorization*

Description

Return the NMF decomposition.

Usage

```
NMF(x, rank = 2, nstart = 10, ...)
```

Arguments

x	A numeric dataset (data.frame or matrix).
rank	Specification of the factorization rank.
nstart	How many random sets should be chosen?
...	Other parameters.

See Also

nmf

Examples

```
## Not run:
install.packages ("BiocManager")
BiocManager::install ("Biobase")
install.packages ("NMF")
require (datasets)
data (iris)
NMF (iris [, -5])

## End(Not run)
```

---

ozone *Ozone dataset*

Description

This dataset constains measurements on ozone level.

Usage

```
ozone
```

Format

Each instance is described by the maximum level of ozone measured during the day. Temperature, clouds, and wind are also recorded.

Source

https://r-stat-sc-donnees.github.io/ozone.txt

params-class *Learning Parameters*

Description

This class contains main parameters for various learning methods.

Slots

`decay` The decay parameter.

`hidden` The number of hidden nodes.

`epsilon` The epsilon parameter.

`gamma` The gamma parameter.

`cost` The cost parameter.

See Also

MLP, MLPREG, SVM, SVR

PCA *Principal Component Analysis (PCA)*

Description

Performs Principal Component Analysis (PCA) with supplementary individuals, supplementary quantitative variables and supplementary categorical variables. Missing values are replaced by the column mean.

Usage

```
PCA(
  d,
  scale.unit = TRUE,
  ncp = ncol(d) - length(quanti.sup) - length(quali.sup),
  ind.sup = NULL,
  quanti.sup = NULL,
  quali.sup = NULL,
  row.w = NULL,
  col.w = NULL
)
```

Arguments

d	A data frame with n rows (individuals) and p columns (numeric variables).
scale.unit	A boolean, if TRUE (value set by default) then data are scaled to unit variance.
ncp	The number of dimensions kept in the results (by default 5).
ind.sup	A vector indicating the indexes of the supplementary individuals.
quanti.sup	A vector indicating the indexes of the quantitative supplementary variables.
quali.sup	A vector indicating the indexes of the categorical supplementary variables.
row.w	An optional row weights (by default, a vector of 1 for uniform row weights); the weights are given only for the active individuals.
col.w	An optional column weights (by default, uniform column weights); the weights are given only for the active variables.

Value

The PCA on the dataset.

See Also

PCA, CA, MCA, plot.factorial, kaiser, factorial-class

Examples

```
require (datasets)
data (iris)
PCA (iris, quali.sup = 5)
```

---

performance *Performance estimation*

Description

Estimate the performance of classification or regression methods using bootstrap or crossvalidation (accuracy, ROC curves, confusion matrices, ...)

Usage

```
performance(
  methods,
  train.x,
  train.y,
  test.x = NULL,
  test.y = NULL,
  train.size = round(0.7 * nrow(train.x)),
  type = c("evaluation", "confusion", "roc", "cost", "scatter", "avsp"),
  protocol = c("bootstrap", "crossvalidation", "loocv", "holdout", "train"),
```

```
  eval = ifelse(is.factor(train.y), "accuracy", "r2"),
  nruns = 10,
  nfolds = 10,
  new = TRUE,
  lty = 1,
  seed = NULL,
  methodparameters = NULL,
  names = NULL,
  ...
)
```

Arguments

methods	The classification or regression methods to be evaluated.
train.x	The dataset (description/predictors), a matrix or data.frame.
train.y	The target (class labels or numeric values), a factor or vector.
test.x	The test dataset (description/predictors), a matrix or data.frame.
test.y	The (test) target (class labels or numeric values), a factor or vector.
train.size	The size of the training set (holdout estimation).
type	The type of evaluation (confusion matrix, ROC curve, ...)
protocol	The evaluation protocol (crossvalidation, bootstrap, ...)
eval	The evaluation functions.
nruns	The number of bootstrap runs.
nfolds	The number of folds (crossvalidation estimation).
new	A logical value indicating whether a new plot should be be created or not (cost curves or ROC curves).
lty	The line type (and color) specified as an integer (cost curves or ROC curves).
seed	A specified seed for random number generation (useful for testing different method with the same bootstap samplings).
methodparameters	Method parameters (if null tuning is done by cross-validation).
names	Method names.
...	Other specific parameters for the leaning method.

Value

The evaluation of the predictions (numeric value).

See Also

confusion, evaluation, cost.curves, roc.curves

Examples

```
## Not run:
require ("datasets")
data (iris)
# One method, one evaluation criterion, bootstrap estimation
performance (NB, iris [, -5], iris [, 5], seed = 0)
# One method, two evaluation criteria, train set estimation
performance (NB, iris [, -5], iris [, 5], eval = c ("accuracy", "kappa"),
             protocol = "train", seed = 0)
# Three methods, ROC curves, LOOCV estimation
performance (c (NB, LDA, LR), linsep [, -3], linsep [, 3], type = "roc",
             protocol = "loocv", seed = 0)
# List of methods in a variable, confusion matrix, hodout estimation
classif = c (NB, LDA, LR)
performance (classif, iris [, -5], iris [, 5], type = "confusion",
             protocol = "holdout", seed = 0, names = c ("NB", "LDA", "LR"))
# List of strings (method names), scatterplot evaluation, crossvalidation estimation
classif = c ("NB", "LDA", "LR")
performance (classif, iris [, -5], iris [, 5], type = "scatter",
             protocol = "crossvalidation", seed = 0)
# Actual vs. predicted
data (trees)
performance (LINREG, trees [, -3], trees [, 3], type = "avsp")

## End(Not run)
```

---

plot.cda *Plot function for cda-class*

Description

Plot the learning set (and test set) on the canonical axes obtained by Canonical Discriminant Analysis (function CDA).

Usage

```
## S3 method for class 'cda'
plot(x, newdata = NULL, axes = 1:2, ...)
```

Arguments

x	The classification model (object of class cda-class).
newdata	The test set (matrix or data.frame).
axes	The canonical axes to be printed (numeric vector).
...	Other parameters.

See Also

CDA, predict.cda, cda-class

Examples

```
require (datasets)
data (iris)
model = CDA (iris [, -5], iris [, 5])
plot (model)
```

`plot.factorial` *Plot function for factorial-class*

Description

Plot PCA, CA or MCA.

Usage

```
## S3 method for class 'factorial'
plot(x, type = c("ind", "cor", "eig"), axes = c(1, 2), ...)
```

Arguments

`x`	The PCA, CA or MCA result (object of class `factorial-class`).
`type`	The graph to plot.
`axes`	The factorial axes to be printed (numeric `vector`).
`...`	Other parameters.

See Also

`CA`, `MCA`, `PCA`, `plot.CA`, `plot.MCA`, `plot.PCA`, `factorial-class`

Examples

```
require (datasets)
data (iris)
pca = PCA (iris, quali.sup = 5)
plot (pca)
plot (pca, type = "cor")
plot (pca, type = "eig")
```

plot.som *Plot function for som-class*

Description

Plot Kohonen's self-organizing maps.

Usage

```
## S3 method for class 'som'
plot(x, type = c("scatter", "mapping"), col = NULL, labels = FALSE, ...)
```

Arguments

x	The Kohonen's map (object of class som-class).
type	The type of plot.
col	Color of the data points
labels	A vector of character strings to be printed instead of points in the plot.
...	Other parameters.

See Also

SOM, som-class

Examples

```
require (datasets)
data (iris)
som = SOM (iris [, -5], xdim = 5, ydim = 5, post = "ward", k = 3)
plot (som) # Scatter plot (default)
plot (som, type = "mapping") # Kohonen map
```

plotavsp *Plot actual vs. predictions*

Description

Plot actual vs. predictions of a regression model.

Usage

```
plotavsp(predictions, gt)
```

Arguments

predictions	The predictions of a classification model (vector).
gt	The ground truth of the dataset (vector).

See Also

confusion, evaluation.accuracy, evaluation.fmeasure, evaluation.fowlkesmallows, evaluation.goodness, evaluation.jaccard, evaluation.kappa, evaluation.precision, evaluation.recall, evaluation.msep, evaluation.r2, performance

Examples

```
require (datasets)
data (trees)
model = LINREG (trees [, -3], trees [, 3])
pred = predict (model, trees [, -3])
plotavsp (pred, trees [, 3])
```

---

plotcloud *Plot word cloud*

Description

Plot a word cloud based on the word frequencies in the documents.

Usage

```
plotcloud(corpus, k = NULL, stopwords = "en", ...)
```

Arguments

corpus	The corpus of documents (a vector of characters) or the vocabulary of the documents (result of function getvocab).
k	A categorial variable (vector or factor).
stopwords	Stopwords, or the language of the documents. NULL if stop words should not be removed.
...	Other parameters.

See Also

plotzipf, getvocab, wordcloud

Examples

```
## Not run:
text = loadtext ("http://mattmahoney.net/dc/text8.zip")
plotcloud (text)
vocab = getvocab (text, mincount = 1, lang = NULL, stopwords = "en")
plotcloud (vocab)

## End(Not run)
```

`plotclus` *Generic Plot Method for Clustering*

Description

Plot a clustering according to various parameters

Usage

```
plotclus(
  clustering,
  d = NULL,
  type = c("scatter", "boxplot", "tree", "height", "mapping", "words"),
  centers = FALSE,
  k = NULL,
  tailsize = 9,
  ...
)
```

Arguments

`clustering`	The clustering to be plotted.
`d`	The dataset (`matrix` or `data.frame`), mandatory for some of the graphics.
`type`	The type of plot.
`centers`	Indicates whether or not cluster centers should be plotted (used only in scatter plots).
`k`	Number of clusters (used only for hierarchical methods). If not specified an "optimal" value is determined.
`tailsize`	Number of clusters showned (used only for height plots).
`...`	Other parameters.

See Also

`treeplot`, `scatterplot`, `plot.som`, `boxclus`

Examples

```
## Not run:
require (datasets)
data (iris)
ward = HCA (iris [, -5], method = "ward", k = 3)
plotclus (ward, iris [, -5], type = "scatter") # Scatter plot
plotclus (ward, iris [, -5], type = "boxplot") # Boxplot
plotclus (ward, iris [, -5], type = "tree") # Dendrogram
plotclus (ward, iris [, -5], type = "height") # Distances between merging clusters
som = SOM (iris [, -5], xdim = 5, ydim = 5, post = "ward", k = 3)
plotclus (som, iris [, -5], type = "scatter") # Scatter plot for SOM
plotclus (som, iris [, -5], type = "mapping") # Kohonen map

## End(Not run)
```

---

plotdata *Advanced plot function*

Description

Plot a dataset.

Usage

```
plotdata(
  d,
  k = NULL,
 type = c("pairs", "scatter", "parallel", "boxplot", "histogram", "barplot", "pie",
    "heatmap", "heatmapc", "pca", "cda", "svd", "nmf", "tsne", "som", "words"),
  legendpos = "topleft",
  alpha = 200,
  asp = 1,
  labels = FALSE,
  ...
)
```

Arguments

d	A numeric dataset (data.frame or matrix).
k	A categorial variable (vector or factor).
type	The type of graphic to be plotted.
legendpos	Position of the legend
alpha	Color opacity (0-255).
asp	Aspect ratio (default: 1).
labels	Indicates whether or not labels (row names) should be showned on the (scatter) plot.
...	Other parameters.

Examples

```
require (datasets)
data (iris)
# Without classification
plotdata (iris [, -5]) # Défault (pairs)
# With classification
plotdata (iris [, -5], iris [, 5]) # Défault (pairs)
plotdata (iris, 5) # Column number
plotdata (iris) # Automatic detection of the classification (if only one factor column)
plotdata (iris, type = "scatter") # Scatter plot (PCA axis)
plotdata (iris, type = "parallel") # Parallel coordinates
plotdata (iris, type = "boxplot") # Boxplot
plotdata (iris, type = "histogram") # Histograms
plotdata (iris, type = "heatmap") # Heatmap
plotdata (iris, type = "heatmapc") # Heatmap (and hierarchalcal clustering)
plotdata (iris, type = "pca") # Scatter plot (PCA axis)
plotdata (iris, type = "cda") # Scatter plot (CDA axis)
plotdata (iris, type = "svd") # Scatter plot (SVD axis)
plotdata (iris, type = "som") # Kohonen map
# With only one variable
plotdata (iris [, 1], iris [, 5]) # Défault (data vs. index)
plotdata (iris [, 1], iris [, 5], type = "scatter") # Scatter plot (data vs. index)
plotdata (iris [, 1], iris [, 5], type = "boxplot") # Boxplot
# With two variables
plotdata (iris [, 3:4], iris [, 5]) # Défault (scatter plot)
plotdata (iris [, 3:4], iris [, 5], type = "scatter") # Scatter plot
data (titanic)
plotdata (titanic, type = "barplot") # Barplots
plotdata (titanic, type = "pie") # Pie charts
```

---

plotzipf *Plot rank versus frequency*

Description

Plot the frequency of words in a document agains the ranks of those words. It also plot the Zipf law.

Usage

```
plotzipf(corpus)
```

Arguments

corpus	The corpus of documents (a vector of characters) or the vocabulary of the documents (result of function getvocab).

See Also

plotcloud, getvocab

Examples

```
## Not run:
text = loadtext ("http://mattmahoney.net/dc/text8.zip")
plotzipf (text)
vocab = getvocab (text, mincount = 1, lang = NULL)
plotzipf (vocab)

## End(Not run)
```

POLYREG *Polynomial Regression*

Description

This function builds a polynomial regression model.

Usage

```
POLYREG(x, y, degree = 2, tune = FALSE, ...)
```

Arguments

x	Predictor matrix.
y	Response vector.
degree	The polynom degree.
tune	If true, the function returns paramters instead of a classification model.
...	Other parameters.

Value

The classification model, as an object of class model-class.

See Also

polyreg

Examples

```
## Not run:
require (datasets)
data (trees)
POLYREG (trees [, -3], trees [, 3])

## End(Not run)
```

predict.apriori *Model predictions*

Description

This function predicts values based upon a model trained by `apriori.classif`. Observations that do not match any of the rules are labelled as "unmatched".

Usage

```
## S3 method for class 'apriori'
predict(object, test, unmatched = "Unknown", ...)
```

Arguments

`object`	The classification model (of class `apriori`, created by `apriori.classif`).
`test`	The test set (a `data.frame`)
`unmatched`	The class label given to the unmatched observations (a character string).
`...`	Other parameters.

Value

A vector of predicted values (`factor`).

See Also

APRIORI, apriori-class, apriori

Examples

```
require ("datasets")
data (iris)
d = discretizeDF (iris,
   default = list (method = "interval", breaks = 3, labels = c ("small", "medium", "large")))
model = APRIORI (d [, -5], d [, 5], supp = .1, conf = .9, prune = TRUE)
predict (model, d [, -5])
```

predict.boosting *Model predictions*

Description

This function predicts values based upon a model trained by a boosting method.

Usage

```
## S3 method for class 'boosting'
predict(object, test, fuzzy = FALSE, ...)
```

Arguments

object	The classification model (of class boosting-class, created by ADABOOST or BAGGING).
test	The test set (a data.frame)
fuzzy	A boolean indicating whether fuzzy classification is used or not.
...	Other parameters.

Value

A vector of predicted values (factor).

See Also

ADABOOST, BAGGING, boosting-class

Examples

```
## Not run:
require (datasets)
data (iris)
d = splitdata (iris, 5)
model = BAGGING (d$train.x, d$train.y, NB)
predict (model, d$test.x)
model = ADABOOST (d$train.x, d$train.y, NB)
predict (model, d$test.x)

## End(Not run)
```

predict.cda *Model predictions*

Description

This function predicts values based upon a model trained by CDA.

Usage

```
## S3 method for class 'cda'
predict(object, test, fuzzy = FALSE, ...)
```

Arguments

object	The classification model (of class cda-class, created by CDA).
test	The test set (a data.frame)
fuzzy	A boolean indicating whether fuzzy classification is used or not.
...	Other parameters.

Value

A vector of predicted values (factor).

See Also

CDA, plot.cda, cda-class

Examples

```
require (datasets)
data (iris)
d = splitdata (iris, 5)
model = CDA (d$train.x, d$train.y)
predict (model, d$test.x)
```

predict.dbs *Predict function for DBSCAN*

Description

Return the closest DBSCAN cluster for a new dataset.

Usage

```
## S3 method for class 'dbs'
predict(object, newdata, ...)
```

Arguments

object	The classification model (of class dbs-class, created by DBSCAN).
newdata	A new dataset (a data.frame), with same variables as the learning dataset.
...	Other parameters.

See Also

DBSCAN

Examples

```
require (datasets)
data (iris)
d = splitdata (iris, 5)
model = DBSCAN (d$train.x, minpts = 5, eps = 0.65)
predict (model, d$test.x)
```

---

predict.em *Predict function for EM*

Description

Return the closest EM cluster for a new dataset.

Usage

```
## S3 method for class 'em'
predict(object, newdata, ...)
```

Arguments

object	The classification model (of class em-class, created by EM).
newdata	A new dataset (a data.frame), with same variables as the learning dataset.
...	Other parameters.

See Also

EM

Examples

```
require (datasets)
data (iris)
d = splitdata (iris, 5)
model = EM (d$train.x, 3)
predict (model, d$test.x)
```

predict.kmeans *Predict function for K-means*

Description

Return the closest K-means cluster for a new dataset.

Usage

```
## S3 method for class 'kmeans'
predict(object, newdata, ...)
```

Arguments

object	The classification model (created by KMEANS).
newdata	A new dataset (a data.frame), with same variables as the learning dataset.
...	Other parameters.

See Also

KMEANS

Examples

```
require (datasets)
data (iris)
d = splitdata (iris, 5)
model = KMEANS (d$train.x, k = 3)
predict (model, d$test.x)
```

predict.knn *Model predictions*

Description

This function predicts values based upon a model trained by KNN.

Usage

```
## S3 method for class 'knn'
predict(object, test, fuzzy = FALSE, ...)
```

Arguments

object	The classification model (of class knn).
test	The test set (a data.frame).
fuzzy	A boolean indicating whether fuzzy classification is used or not.
...	Other parameters.

Value

A vector of predicted values (factor).

See Also

KNN, knn-class

Examples

```
require (datasets)
data (iris)
d = splitdata (iris, 5)
model = KNN (d$train.x, d$train.y)
predict (model, d$test.x)
```

---

predict.meanshift *Predict function for MeanShift*

Description

Return the closest MeanShift cluster for a new dataset.

Usage

```
## S3 method for class 'meanshift'
predict(object, newdata, ...)
```

Arguments

object	The classification model (created by MEANSHIFT).
newdata	A new dataset (a data.frame), with same variables as the learning dataset.
...	Other parameters.

See Also

MEANSHIFT

Examples

```
## Not run:
require (datasets)
data (iris)
d = splitdata (iris, 5)
model = MEANSHIFT (d$train.x, bandwidth = .75)
predict (model, d$test.x)

## End(Not run)
```

`predict.model` *Model predictions*

Description

This function predicts values based upon a model trained by any classification or regression model.

Usage

```
## S3 method for class 'model'
predict(object, test, fuzzy = FALSE, ...)
```

Arguments

`object`	The classification model (of class `cda-class`, created by `CDA`).
`test`	The test set (a `data.frame`).
`fuzzy`	A boolean indicating whether fuzzy classification is used or not.
`...`	Other parameters.

Value

A vector of predicted values (`factor`).

See Also

`model-class`

Examples

```
require (datasets)
data (iris)
d = splitdata (iris, 5)
model = LDA (d$train.x, d$train.y)
predict (model, d$test.x)
```

predict.selection *Model predictions*

Description

This function predicts values based upon a model trained by any classification or regression model.

Usage

```
## S3 method for class 'selection'
predict(object, test, fuzzy = FALSE, ...)
```

Arguments

object	The classification model (of class cda-class, created by CDA).
test	The test set (a data.frame).
fuzzy	A boolean indicating whether fuzzy classification is used or not.
...	Other parameters.

Value

A vector of predicted values (factor).

See Also

FEATURESELECTION, selection-class

Examples

```
## Not run:
require (datasets)
data (iris)
d = splitdata (iris, 5)
model = FEATURESELECTION (d$train.x, d$train.y, uninb = 2, mainmethod = LDA)
predict (model, d$test.x)

## End(Not run)
```

`predict.textmining` *Model predictions*

Description

This function predicts values based upon a model trained for text mining.

Usage

```
## S3 method for class 'textmining'
predict(object, test, fuzzy = FALSE, ...)
```

Arguments

`object`	The classification model (of class `textmining-class`, created by `TEXTMINING`.
`test`	The test set (a `data.frame`)
`fuzzy`	A boolean indicating whether fuzzy classification is used or not.
`...`	Other parameters.

Value

A vector of predicted values (`factor`).

See Also

`TEXTMINING`, `textmining-class`

Examples

```
## Not run:
require (text2vec)
data ("movie_review")
d = movie_review [, 2:3]
d [, 1] = factor (d [, 1])
d = splitdata (d, 1)
model = TEXTMINING (d$train.x, NB, labels = d$train.y, mincount = 50)
pred = predict (model, d$test.x)
evaluation (pred, d$test.y)

## End(Not run)
```

print.apriori *Print a classification model obtained by APRIORI*

Description

Print the set of rules in the classification model.

Usage

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

Arguments

x	The model to be printed.
...	Other parameters.

See Also

APRIORI, predict.apriori, summary.apriori, apriori-class, apriori

Examples

```
require ("datasets")
data (iris)
d = discretizeDF (iris,
   default = list (method = "interval", breaks = 3, labels = c ("small", "medium", "large")))
model = APRIORI (d [, -5], d [, 5], supp = .1, conf = .9, prune = TRUE)
print (model)
```

print.factorial *Plot function for factorial-class*

Description

Print PCA, CA or MCA.

Usage

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

Arguments

x	The PCA, CA or MCA result (object of class factorial-class).
...	Other parameters.

See Also

CA, MCA, PCA, print.CA, print.MCA, print.PCA, factorial-class

Examples

```
require (datasets)
data (iris)
pca = PCA (iris, quali.sup = 5)
print (pca)
```

---

pseudoF *Pseudo-F*

Description

Compute the pseudo-F of a clustering result obtained by the K-means method.

Usage

```
pseudoF(clustering)
```

Arguments

clustering	The clustering result (obtained by the function kmeans).

Value

The pseudo-F of the clustering result.

See Also

kmeans.getk, KMEANS, kmeans

Examples

```
require (datasets)
data (iris)
km = KMEANS (iris [, -5], k = 3)
pseudoF (km)
```

QDA *Classification using Quadratic Discriminant Analysis*

Description

This function builds a classification model using Quadratic Discriminant Analysis.

Usage

```
QDA(train, labels, tune = FALSE, ...)
```

Arguments

train	The training set (description), as a data.frame.
labels	Class labels of the training set (vector or factor).
tune	If true, the function returns paramters instead of a classification model.
...	Other parameters.

Value

The classification model.

See Also

qda

Examples

```
require (datasets)
data (iris)
QDA (iris [, -5], iris [, 5])
```

query.docs *Document query*

Description

Search for documents similar to the query.

Usage

```
query.docs(docvectors, query, vectorizer, nres = 5)
```

Arguments

docvectors	The vectorized documents.
query	The query (vectorized or raw text).
vectorizer	The vectorizer taht has been used to vectorize the documents.
nres	The number of results.

Value

The indices of the documents the most similar to the query.

See Also

vectorize.docs, sim2

Examples

```
## Not run:
require (text2vec)
data (movie_review)
vectorizer = vectorize.docs (corpus = movie_review$review,
                             minphrasecount = 50, returndata = FALSE)
docs = vectorize.docs (corpus = movie_review$review, vectorizer = vectorizer)
query.docs (docs, movie_review$review [1], vectorizer)
query.docs (docs, docs [1, ], vectorizer)

## End(Not run)
```

---

query.words *Word query*

Description

Search for words similar to the query.

Usage

```
query.words(wordvectors, origin, sub = NULL, add = NULL, nres = 5, lang = "en")
```

Arguments

wordvectors	The vectorized words
origin	The query (character).
sub	Words to be substrated to the origin.
add	Words to be Added to the origin.
nres	The number of results.
lang	The language of the words (NULL if no stemming).

Value

The Words the most similar to the query.

See Also

vectorize.words, sim2

Examples

```
## Not run:
text = loadtext ("http://mattmahoney.net/dc/text8.zip")
words = vectorize.words (text, minphrasecount = 50)
query.words (words, origin = "paris", sub = "france", add = "germany")
query.words (words, origin = "berlin", sub = "germany", add = "france")
query.words (words, origin = "new_zealand")

## End(Not run)
```

RANDOMFOREST *Classification using Random Forest*

Description

This function builds a classification model using Random Forest

Usage

```
RANDOMFOREST(
  train,
  labels,
  ntree = 500,
 nvar = if (!is.null(labels) && !is.factor(labels)) max(floor(ncol(train)/3), 1) else
    floor(sqrt(ncol(train))),
  tune = FALSE,
  ...
)
```

Arguments

train	The training set (description), as a data.frame.
labels	Class labels of the training set (vector or factor).
ntree	The number of trees in the forest.
nvar	Number of variables randomly sampled as candidates at each split.
tune	If true, the function returns paramters instead of a classification model.
...	Other parameters.

Value

The classification model.

See Also

randomForest

Examples

```
## Not run:
require (datasets)
data (iris)
RANDOMFOREST (iris [, -5], iris [, 5])

## End(Not run)
```

---

reg1 *reg1 dataset*

Description

Artificial dataset for simple regression tasks.

Usage

```
reg1
reg1.train
reg1.test
```

Format

50 instances and 3 variables. X, a numeric, K, a factor, and Y, a numeric (the target variable).

Author(s)

Alexandre Blansché <alexandre.blansche@univ-lorraine.fr>

reg2 *reg2 dataset*

Description

Artificial dataset for simple regression tasks.

Usage

```
reg2
reg2.train
reg2.test
```

Format

50 instances and 2 variables. `X` and `Y` (the target variable) are both numeric variables.

Author(s)

Alexandre Blansché <alexandre.blansche@univ-lorraine.fr>

regplot *Plot function for a regression model*

Description

Plot a regresion model on a 2-D plot. The predictor `x` should be one-dimensional.

Usage

```
regplot(model, x, y, margin = 0.1, ...)
```

Arguments

`model`	The model to be plotted.
`x`	The predictor `vector`.
`y`	The response `vector`.
`margin`	A margin parameter.
`...`	Other graphical parameters

Examples

```
require (datasets)
data (cars)
model = POLYREG (cars [, -2], cars [, 2])
regplot (model, cars [, -2], cars [, 2])
```

resplot *Plot the studentized residuals of a linear regression model*

Description

Plot the studentized residuals of a linear regression model.

Usage

```
resplot(model, index = NULL, labels = NULL)
```

Arguments

model	The model to be plotted.
index	The index of the variable used for for the x-axis.
labels	The labels of the instances.

Examples

```
require (datasets)
data (trees)
model = LINREG (trees [, -3], trees [, 3])
resplot (model) # Ordered by index
resplot (model, index = 0) # Ordered by variable "Volume" (dependant variable)
resplot (model, index = 1) # Ordered by variable "Girth" (independant variable)
resplot (model, index = 2) # Ordered by variable "Height" (independant variable)
```

roc.curves *Plot ROC Curves*

Description

This function plots ROC Curves of several classification predictions.

Usage

```
roc.curves(predictions, gt, methods.names = NULL)
```

Arguments

predictions	The predictions of a classification model (factor or vector).
gt	Actual labels of the dataset (factor or vector).
methods.names	The name of the compared methods (vector).

Value

The evaluation of the predictions (numeric value).

See Also

cost.curves, performance

Examples

```
require (datasets)
data (iris)
d = iris
levels (d [, 5]) = c ("+", "+", "-") # Building a two classes dataset
model.nb = NB (d [, -5], d [, 5])
model.lda = LDA (d [, -5], d [, 5])
pred.nb = predict (model.nb, d [, -5])
pred.lda = predict (model.lda, d [, -5])
roc.curves (cbind (pred.nb, pred.lda), d [, 5], c ("NB", "LDA"))
```

---

rotation *Rotation*

Description

Rotation on two variables of a numeric dataset

Usage

```
rotation(d, angle, axis = 1:2, range = 2 * pi)
```

Arguments

d	The dataset.
angle	The angle of the rotation.
axis	The axis.
range	The range of the angle (360, 2*pi, 100, ...)

Value

A rotated data matrix.

Examples

```
d = data.parabol ()
d [, -3] = rotation (d [, -3], 45, range = 360)
plotdata (d [, -3], d [, 3])
```

runningtime *Running time*

Description

Return the running time of a function

Usage

```
runningtime(FUN, ...)
```

Arguments

FUN	The function to be evaluated.
...	The parameters to be passes to function FUN.

Value

The running time of function FUN.

See Also

difftime

Examples

```
sqrt (x = 1:100)
runningtime (sqrt, x = 1:100)
```

scatterplot *Clustering Scatter Plots*

Description

Produce a scatter plot for clustering results. If the dataset has more than two dimensions, the scatter plot will show the two first PCA axes.

Usage

```
scatterplot(
  d,
  clusters,
  centers = NULL,
  labels = FALSE,
  ellipses = FALSE,
  legend = c("auto1", "auto2"),
  ...
)
```

Arguments

d	The dataset (matrix or data.frame).
clusters	Cluster labels of the training set (vector or factor).
centers	Coordinates of the cluster centers.
labels	Indicates whether or not labels (row names) should be showned on the plot.
ellipses	Indicates whether or not ellipses should be drawned around clusters.
legend	Indicates where the legend is placed on the graphics.
...	Other parameters.

Examples

```
require (datasets)
data (iris)
km = KMEANS (iris [, -5], k = 3)
scatterplot (iris [, -5], km$cluster)
```

---

selectfeatures *Feature selection for classification*

Description

Select a subset of features for a classification task.

Usage

```
selectfeatures(
  train,
  labels,
  algorithm = c("ranking", "forward", "backward", "exhaustive"),
 unieval = if (algorithm[1] == "ranking") c("fisher", "fstat", "relief", "inertiaratio")
    else NULL,
  uninb = NULL,
  unithreshold = NULL,
  multieval = if (algorithm[1] == "ranking") NULL else c("mrmr", "cfs", "fstat",
    "inertiaratio", "wrapper"),
  wrapmethod = NULL,
  keep = FALSE,
  ...
)
```

Arguments

train	The training set (description), as a data.frame.
labels	Class labels of the training set (vector or factor).
algorithm	The feature selection algorithm.
unieval	The (univariate) evaluation criterion. uninb, unithreshold or multieval must be specified.
uninb	The number of selected feature (univariate evaluation).
unithreshold	The threshold for selecting feature (univariate evaluation).
multieval	The (multivariate) evaluation criterion.
wrapmethod	The classification method used for the wrapper evaluation.
keep	If true, the dataset is kept in the returned result.
...	Other parameters.

See Also

FEATURESELECTION, selection-class

Examples

```
## Not run:
require (datasets)
data (iris)
selectfeatures (iris [, -5], iris [, 5], algorithm = "forward", multieval = "fstat")
selectfeatures (iris [, -5], iris [, 5], algorithm = "ranking", uninb = 2)
selectfeatures (iris [, -5], iris [, 5], algorithm = "ranking",
                multieval = "wrapper", wrapmethod = LDA)

## End(Not run)
```

---

selection-class *Feature selection*

Description

This class contains the result of feature selection algorithms.

Slots

selection A vector of integers indicating the selected features.

unieval The evaluation of the features (univariate).

multieval The evaluation of the selected features (multivariate).

algorithm The algorithm used to select features.

univariate The evaluation criterion (univariate).

`nbfeatures` The number of features to be kept.

`threshold` The threshold to decide whether a feature is kept or not..

`multivariate` The evaluation criterion (multivariate).

`dataset` The dataset described by the selected features only.

`model` The classification model.

See Also

`FEATURESELECTION`, `predict.selection`, `selectfeatures`

`snore` *Snore dataset*

Description

This dataset has been used in a study on snoring in Angers hospital.

Usage

```
snore
```

Format

The dataset has 100 instances described by 7 variables. The variables are as follows:

`Age` In years.

`Weights` In kg.

`Height` In cm.

`Alcool` Number of glass of alcool per day.

`Sex` M for male or F for female.

`Snore` Snoring diagnosis (Y or N).

`Tobacco` Y or N.

Source

http://forge.info.univ-angers.fr/~gh/Datasets/datasets.htm

SOM *Self-Organizing Maps clustering method*

Description

Run the SOM algorithm for clustering.

Usage

```
SOM(
  d,
  xdim = floor(sqrt(nrow(d))),
  ydim = floor(sqrt(nrow(d))),
  rlen = 10000,
  post = c("none", "single", "ward"),
  k = NULL,
  ...
)
```

Arguments

d	The dataset (matrix or data.frame).
xdim, ydim	The dimensions of the grid.
rlen	The number of iterations.
post	The post-treatement method: "none" (None), "single" (Single link) or "ward" (Ward clustering).
k	The number of cluster (only used if post is different from "none").
...	Other parameters.

Value

The fitted Kohonen's map as an object of class som.

See Also

plot.som, som-class, som

Examples

```
require (datasets)
data (iris)
SOM (iris [, -5], xdim = 5, ydim = 5, post = "ward", k = 3)
```

som-class *Self-Organizing Maps model*

Description

This class contains the model obtained by the SOM method.

Slots

`som` An object of class kohonen representing the fitted map.

`nodes` A `vector` of integer indicating the cluster to which each node is allocated.

`cluster` A `vector` of integer indicating the cluster to which each observation is allocated.

`data` The dataset that has been used to fit the map (as a `matrix`).

See Also

`plot.som`, `SOM`, `som`

SPECTRAL *Spectral clustering method*

Description

Run a Spectral clustering algorithm.

Usage

```
SPECTRAL(d, k, sigma = 1, graph = TRUE, ...)
```

Arguments

`d`	The dataset (`matrix` or `data.frame`).
`k`	The number of cluster.
`sigma`	Width of the gaussian used to build the affinity matrix.
`graph`	A logical indicating whether or not a graphic should be plotted (projection on the spectral space of the affinity matrix).
`...`	Other parameters.

See Also

`spectral-class`

Examples

```
## Not run:
require (datasets)
data (iris)
SPECTRAL (iris [, -5], k = 3)

## End(Not run)
```

`spectral-class` *Spectral clustering model*

Description

This class contains the model obtained by Spectral clustering.

Slots

`cluster` A `vector` of integer indicating the cluster to which each observation is allocated.

`proj` The projection of the dataset in the spectral space.

`centers` The cluster centers (on the spectral space).

See Also

SPECTRAL

`spine` *Spine dataset*

Description

The data have been organized in two different but related classification tasks. The first task consists in classifying patients as belonging to one out of three categories: Normal, Disk Hernia or Spondylolisthesis. For the second task, the categories Disk Hernia and Spondylolisthesis were merged into a single category labelled as 'abnormal'. Thus, the second task consists in classifying patients as belonging to one out of two categories: Normal or Abnormal.

Usage

```
spine
spine.train
spine.test
```

Format

The dataset has 310 instances described by 8 variables. Variables V1 to V6 are biomechanical attributes derived from the shape and orientation of the pelvis and lumbar spine. The variable Classif2 is the classification into two classes AB and NO. The variable Classif3 is the classification into 3 classes DH, SL and NO. spine.train contains 217 instances and spine.test contains 93.

Source

http://archive.ics.uci.edu/ml/datasets/vertebral+column

---

splitdata *Splits a dataset into training set and test set*

Description

This function splits a dataset into training set and test set. Return an object of class dataset-class.

Usage

```
splitdata(dataset, target, size = round(0.7 * nrow(dataset)), seed = NULL)
```

Arguments

dataset	The dataset to be split (data.frame or matrix).
target	The column index of the target variable (class label or response variable).
size	The size of the training set (as an integer value).
seed	A specified seed for random number generation.

Value

An object of class dataset-class.

See Also

dataset-class

Examples

```
require (datasets)
data (iris)
d = splitdata (iris, 5)
str (d)
```

`stability` *Clustering evaluation through stability*

Description

Evaluation a clustering algorithm according to stability, through a bootstrap procedure.

Usage

```
stability(
  clusteringmethods,
  d,
  originals = NULL,
  eval = "jaccard",
  type = c("cluster", "global"),
  nsampling = 10,
  seed = NULL,
  names = NULL,
  graph = FALSE,
  ...
)
```

Arguments

`clusteringmethods`	The clustering methods to be evaluated.
`d`	The dataset.
`originals`	The original clustering.
`eval`	The evaluation criteria.
`type`	The comparison method.
`nsampling`	The number of bootstrap runs.
`seed`	A specified seed for random number generation (useful for testing different method with the same bootstap samplings).
`names`	Method names.
`graph`	Indicates wether or not a graphic is potted for each sample.
`...`	Parameters to be passed to the clustering algorithms.

Value

The evaluation of the clustering algorithm(s) (numeric values).

See Also

`compare`, `intern`

Examples

```
## Not run:
require (datasets)
data (iris)
stability (KMEANS, iris [, -5], seed = 0, k = 3)
stability (KMEANS, iris [, -5], seed = 0, k = 3, eval = c ("jaccard", "accuracy"), type = "global")
stability (KMEANS, iris [, -5], seed = 0, k = 3, type = "cluster")
stability (KMEANS, iris [, -5], seed = 0, k = 3, eval = c ("jaccard", "accuracy"), type = "cluster")
stability (c (KMEANS, HCA), iris [, -5], seed = 0, k = 3)
stability (c (KMEANS, HCA), iris [, -5], seed = 0, k = 3,
eval = c ("jaccard", "accuracy"), type = "global")
stability (c (KMEANS, HCA), iris [, -5], seed = 0, k = 3, type = "cluster")
stability (c (KMEANS, HCA), iris [, -5], seed = 0, k = 3,
eval = c ("jaccard", "accuracy"), type = "cluster")
stability (KMEANS, iris [, -5], originals = KMEANS (iris [, -5], k = 3)$cluster, seed = 0, k = 3)
stability (KMEANS, iris [, -5], originals = KMEANS (iris [, -5], k = 3), seed = 0, k = 3)

## End(Not run)
```

STUMP *Classification using one-level decision tree*

Description

This function builds a classification model using CART with maxdepth = 1.

Usage

```
STUMP(train, labels, randomvar = TRUE, tune = FALSE, ...)
```

Arguments

`train`	The training set (description), as a `data.frame`.
`labels`	Class labels of the training set (`vector` or `factor`).
`randomvar`	If true, the model uses a random variable.
`tune`	If true, the function returns paramters instead of a classification model.
`...`	Other parameters.

Value

The classification model.

See Also

CART

Examples

```
require (datasets)
data (iris)
STUMP (iris [, -5], iris [, 5])
```

---

summary.apriori *Print summary of a classification model obtained by APRIORI*

Description

Print summary of the set of rules in the classification model obtained by APRIORI.

Usage

```
## S3 method for class 'apriori'
summary(object, ...)
```

Arguments

object	The model to be printed.
...	Other parameters.

See Also

APRIORI, predict.apriori, print.apriori, apriori-class, apriori

Examples

```
require ("datasets")
data (iris)
d = discretizeDF (iris,
   default = list (method = "interval", breaks = 3, labels = c ("small", "medium", "large")))
model = APRIORI (d [, -5], d [, 5], supp = .1, conf = .9, prune = TRUE)
summary (model)
```

SVD *Singular Value Decomposition*

Description

Return the SVD decomposition.

Usage

```
SVD(x, ndim = min(nrow(x), ncol(x)), ...)
```

Arguments

x	A numeric dataset (data.frame or matrix).
ndim	The number of dimensions.
...	Other parameters.

See Also

svd

Examples

```
require (datasets)
data (iris)
SVD (iris [, -5])
```

SVM *Classification using Support Vector Machine*

Description

This function builds a classification model using Support Vector Machine.

Usage

```
SVM(
  train,
  labels,
  gamma = 2^(-3:3),
  cost = 2^(-3:3),
  kernel = c("radial", "linear"),
  methodparameters = NULL,
  tune = FALSE,
  ...
)
```

Arguments

train	The training set (description), as a data.frame.
labels	Class labels of the training set (vector or factor).
gamma	The gamma parameter (if a vector, cross-over validation is used to chose the best size).
cost	The cost parameter (if a vector, cross-over validation is used to chose the best size).
kernel	The kernel type.

methodparameters	Object containing the parameters. If given, it replaces gamma and cost.
tune	If true, the function returns paramters instead of a classification model.
...	Other arguments.

Value

The classification model.

See Also

svm, SVMl, SVMr

Examples

```
## Not run:
require (datasets)
data (iris)
SVM (iris [, -5], iris [, 5], kernel = "linear", cost = 1)
SVM (iris [, -5], iris [, 5], kernel = "radial", gamma = 1, cost = 1)

## End(Not run)
```

SVMl *Classification using Support Vector Machine with a linear kernel*

Description

This function builds a classification model using Support Vector Machine with a linear kernel.

Usage

```
SVMl(
  train,
  labels,
  cost = 2^(-3:3),
  methodparameters = NULL,
  tune = FALSE,
  ...
)
```

Arguments

train	The training set (description), as a data.frame.
labels	Class labels of the training set (vector or factor).
cost	The cost parameter (if a vector, cross-over validation is used to chose the best size).

methodparameters	Object containing the parameters. If given, it replaces gamma and cost.
tune	If true, the function returns paramters instead of a classification model.
...	Other arguments.

Value

The classification model.

See Also

svm, SVM

Examples

```
## Not run:
require (datasets)
data (iris)
SVMl (iris [, -5], iris [, 5], cost = 1)

## End(Not run)
```

SVMr *Classification using Support Vector Machine with a radial kernel*

Description

This function builds a classification model using Support Vector Machine with a radial kernel.

Usage

```
SVMr(
  train,
  labels,
  gamma = 2^(-3:3),
  cost = 2^(-3:3),
  methodparameters = NULL,
  tune = FALSE,
  ...
)
```

Arguments

train	The training set (description), as a data.frame.
labels	Class labels of the training set (vector or factor).
gamma	The gamma parameter (if a vector, cross-over validation is used to chose the best size).

cost	The cost parameter (if a vector, cross-over validation is used to chose the best size).
methodparameters	Object containing the parameters. If given, it replaces gamma and cost.
tune	If true, the function returns paramters instead of a classification model.
...	Other arguments.

Value

The classification model.

See Also

svm, SVM

Examples

```
## Not run:
require (datasets)
data (iris)
SVMr (iris [, -5], iris [, 5], gamma = 1, cost = 1)

## End(Not run)
```

SVR *Regression using Support Vector Machine*

Description

This function builds a regression model using Support Vector Machine.

Usage

```
SVR(
  x,
  y,
  gamma = 2^(-3:3),
  cost = 2^(-3:3),
  kernel = c("radial", "linear"),
  epsilon = c(0.1, 0.5, 1),
  params = NULL,
  tune = FALSE,
  ...
)
```

Arguments

x	Predictor matrix.
y	Response vector.
gamma	The gamma parameter (if a vector, cross-over validation is used to chose the best size).
cost	The cost parameter (if a vector, cross-over validation is used to chose the best size).
kernel	The kernel type.
epsilon	The epsilon parameter (if a vector, cross-over validation is used to chose the best size).
params	Object containing the parameters. If given, it replaces epsilon, gamma and cost.
tune	If true, the function returns paramters instead of a classification model.
...	Other arguments.

Value

The classification model.

See Also

svm, SVRl, SVRr

Examples

```
## Not run:
require (datasets)
data (trees)
SVR (trees [, -3], trees [, 3], kernel = "linear", cost = 1)
SVR (trees [, -3], trees [, 3], kernel = "radial", gamma = 1, cost = 1)

## End(Not run)
```

SVRl *Regression using Support Vector Machine with a linear kernel*

Description

This function builds a regression model using Support Vector Machine with a linear kernel.

Usage

```
SVRl(
  x,
  y,
  cost = 2^(-3:3),
  epsilon = c(0.1, 0.5, 1),
  params = NULL,
  tune = FALSE,
  ...
)
```

Arguments

x	Predictor matrix.
y	Response vector.
cost	The cost parameter (if a vector, cross-over validation is used to chose the best size).
epsilon	The epsilon parameter (if a vector, cross-over validation is used to chose the best size).
params	Object containing the parameters. If given, it replaces epsilon, gamma and cost.
tune	If true, the function returns paramters instead of a classification model.
...	Other arguments.

Value

The classification model.

See Also

svm, SVR

Examples

```
## Not run:
require (datasets)
data (trees)
SVRl (trees [, -3], trees [, 3], cost = 1)

## End(Not run)
```

`SVRr` *Regression using Support Vector Machine with a radial kernel*

Description

This function builds a regression model using Support Vector Machine with a radial kernel.

Usage

```
SVRr(
  x,
  y,
  gamma = 2^(-3:3),
  cost = 2^(-3:3),
  epsilon = c(0.1, 0.5, 1),
  params = NULL,
  tune = FALSE,
  ...
)
```

Arguments

`x`	Predictor `matrix`.
`y`	Response `vector`.
`gamma`	The gamma parameter (if a vector, cross-over validation is used to chose the best size).
`cost`	The cost parameter (if a vector, cross-over validation is used to chose the best size).
`epsilon`	The epsilon parameter (if a vector, cross-over validation is used to chose the best size).
`params`	Object containing the parameters. If given, it replaces `epsilon`, `gamma` and `cost`.
`tune`	If true, the function returns paramters instead of a classification model.
`...`	Other arguments.

Value

The classification model.

See Also

`svm`, `SVR`

Examples

```
## Not run:
require (datasets)
data (trees)
SVRr (trees [, -3], trees [, 3], gamma = 1, cost = 1)

## End(Not run)
```

`temperature` *Temperature dataset*

Description

The data contains temperature measurement and geographic coordinates of 35 european cities.

Usage

```
temperature
```

Format

The dataset has 35 instances described by 17 variables. Average temperature of the 12 month. Mean and amplitude of the temperature. Latitude and longitude of the city. Localisation in Europe.

`TEXTMINING` *Text mining*

Description

Apply data mining function on vectorized text

Usage

```
TEXTMINING(corpus, miningmethod, vector = c("docs", "words"), ...)
```

Arguments

`corpus`	The corpus.
`miningmethod`	The data mining method.
`vector`	Indicates the type of vectorization, documents (TF-IDF) or words (GloVe).
`...`	Parameters passed to the vectorisation and to the data mining method.

Value

The result of the data mining method.

See Also

predict.textmining, textmining-class, vectorize.docs, vectorize.words

Examples

```
## Not run:
require (text2vec)
data ("movie_review")
d = movie_review [, 2:3]
d [, 1] = factor (d [, 1])
d = splitdata (d, 1)
model = TEXTMINING (d$train.x, NB, labels = d$train.y, mincount = 50)
pred = predict (model, d$test.x)
evaluation (pred, d$test.y)
text = loadtext ("http://mattmahoney.net/dc/text8.zip")
clusters = TEXTMINING (text, HCA, vector = "words", k = 9, maxwords = 100)
plotclus (clusters$res, text, type = "tree", labels = TRUE)

## End(Not run)
```

---

textmining-class *Text mining object*

Description

Object used for text mining.

Slots

`vectorizer` The vectorizer.

`vectors` The vectorized dataset.

`res` The result of the text mining method.

See Also

TEXTMINING, vectorize.docs

titanic *Titanic dataset*

Description

This dataset from the British Board of Trade depict the fate of the passengers and crew during the RMS Titanic disaster.

Usage

```
titanic
```

Format

The dataset has 2201 instances described by 4 variables. The variables are as follows:

`Category` 1st, 2nd, 3rd Class or Crew.

`Age` Adult or Child.

`Sex` Female or Male.

`Fate` Casualty or Survivor.

Source

British Board of Trade (1990), Report on the Loss of the 'Titanic' (S.S.). British Board of Trade Inquiry Report (reprint). Gloucester, UK: Allan Sutton Publishing.

See Also

Titanic

treeplot *Dendrogram Plots*

Description

Draws a dendrogram.

Usage

```
treeplot(
  clustering,
  labels = FALSE,
  k = NULL,
  split = TRUE,
  horiz = FALSE,
  ...
)
```

Arguments

clustering	The dendrogram to be plotted (result of hclust, agnes or HCA).
labels	Indicates whether or not labels (row names) should be showned on the plot.
k	Number of clusters. If not specified an "optimal" value is determined.
split	Indicates wheather or not the clusters should be highlighted in the graphics.
horiz	Indicates if the dendrogram should be drawn horizontally or not.
...	Other parameters.

See Also

dendrogram, HCA, hclust, agnes

Examples

```
require (datasets)
data (iris)
hca = HCA (iris [, -5], method = "ward", k = 3)
treeplot (hca)
```

TSNE *t-distributed Stochastic Neighbor Embedding*

Description

Return the t-SNE dimensionality reduction.

Usage

```
TSNE(x, perplexity = 30, nstart = 10, ...)
```

Arguments

x	A numeric dataset (data.frame or matrix).
perplexity	Specification of the perplexity.
nstart	How many random sets should be chosen?
...	Other parameters.

See Also

Rtsne

Examples

```
require (datasets)
data (iris)
TSNE (iris [, -5])
```

universite *University dataset*

Description

The dataset presents a french university demographics.

Usage

```
universite
```

Format

The dataset has 10 instances (university departments) described by 12 variables. The fist six variables are the number of female and male student studying for bachelor degree (Licence), master degree (Master) and doctorate (Doctorat). The six last variables are obtained by combining the first ones.

Source

https://husson.github.io/data.html

vectorize.docs *Document vectorization*

Description

Vectorize a corpus of documents.

Usage

```
vectorize.docs(
  vectorizer = NULL,
  corpus = NULL,
  lang = "en",
  stopwords = lang,
  ngram = 1,
  mincount = 10,
  minphrasecount = NULL,
  transform = c("tfidf", "lsa", "l1", "none"),
  latentdim = 50,
  returndata = TRUE,
  ...
)
```

Arguments

vectorizer	The document vectorizer.
corpus	The corpus of documents (a vector of characters).
lang	The language of the documents (NULL if no stemming).
stopwords	Stopwords, or the language of the documents. NULL if stop words should not be removed.
ngram	maximum size of n-grams.
mincount	Minimum word count to be considered as frequent.
minphrasecount	Minimum collocation of words count to be considered as frequent.
transform	Transformation (TF-IDF, LSA, L1 normanization, or nothing).
latentdim	Number of latent dimensions if LSA transformation is performed.
returndata	If true, the vectorized documents are returned. If false, a "vectorizer" is returned.
...	Other parameters.

Value

The vectorized documents.

See Also

query.docs, stopwords, vectorizers

Examples

```
## Not run:
require (text2vec)
data ("movie_review")
# Clustering
docs = vectorize.docs (corpus = movie_review$review, transform = "tfidf")
km = KMEANS (docs [sample (nrow (docs), 100), ], k = 10)
# Classification
d = movie_review [, 2:3]
d [, 1] = factor (d [, 1])
d = splitdata (d, 1)
vectorizer = vectorize.docs (corpus = d$train.x,
                             returndata = FALSE, mincount = 50)
train = vectorize.docs (corpus = d$train.x, vectorizer = vectorizer)
test = vectorize.docs (corpus = d$test.x, vectorizer = vectorizer)
model = NB (as.matrix (train), d$train.y)
pred = predict (model, as.matrix (test))
evaluation (pred, d$test.y)

## End(Not run)
```

`vectorize.words` *Word vectorization*

Description

Vectorize words from a corpus of documents.

Usage

```
vectorize.words(
  corpus = NULL,
  ndim = 50,
  maxwords = NULL,
  mincount = 5,
  minphrasecount = NULL,
  window = 5,
  maxcooc = 10,
  maxiter = 10,
  epsilon = 0.01,
  lang = "en",
  stopwords = lang,
  ...
)
```

Arguments

`corpus`	The corpus of documents (a vector of characters).
`ndim`	The number of dimensions of the vector space.
`maxwords`	The maximum number of words.
`mincount`	Minimum word count to be considered as frequent.
`minphrasecount`	Minimum collocation of words count to be considered as frequent.
`window`	Window for term-co-occurence matrix construction.
`maxcooc`	Maximum number of co-occurrences to use in the weighting function.
`maxiter`	The maximum number of iteration to fit the GloVe model.
`epsilon`	Defines early stopping strategy when fit the GloVe model.
`lang`	The language of the documents (NULL if no stemming).
`stopwords`	Stopwords, or the language of the documents. NULL if stop words should not be removed.
`...`	Other parameters.

Value

The vectorized words.

See Also

query.words, stopwords, vectorizers

Examples

```
## Not run:
text = loadtext ("http://mattmahoney.net/dc/text8.zip")
words = vectorize.words (text, minphrasecount = 50)
query.words (words, origin = "paris", sub = "france", add = "germany")
query.words (words, origin = "berlin", sub = "germany", add = "france")
query.words (words, origin = "new_zealand")

## End(Not run)
```

---

vectorizer-class *Document vectorization object*

Description

This class contains a vectorization model for textual documents.

Slots

`vectorizer` The vectorizer.

`transform` The transformation to be applied after vectorization (normalization, TF-IDF).

`phrases` The phrase detection method.

`tfidf` The TF-IDF transformation.

`lsa` The LSA transformation.

`tokens` The token from the original document.

See Also

vectorize.docs, query.docs

vowels *Vowels dataset*

Description

Excerpt of the Letter Recognition Data Set (UCI repository).

Usage

```
vowels
vowels.train
vowels.test
```

Format

The dataset has 4664 instances described by 17 variables. The first variable is the classification into 6 classes (letter A, E, I, O, U and Y). `vowels.train` contains 233 instances and `vowels.test` contains 4431.

Source

https://archive.ics.uci.edu/ml/datasets/letter+recognition

wheat *Wheat dataset*

Description

The data contains kernels belonging to three different varieties of wheat: Kama, Rosa and Canadian, 70 elements each, randomly selected. High quality visualization of the internal kernel structure was detected using a soft X-ray technique. The images were recorded on 13x18 cm X-ray KODAK plates. Source : Institute of Agrophysics of the Polish Academy of Sciences in Lublin.

Usage

```
wheat
```

Format

The dataset has 210 instances described by 8 variables: area, perimeter, compactness, length, width, asymmetry coefficient, groove length and variery.

Source

https://archive.ics.uci.edu/ml/datasets/seeds

wine *Wine dataset*

Description

These data are the results of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. The analysis determined the quantities of 13 constituents found in each of the three types of wines.

Usage

```
wine
```

Format

There are 178 observations and 14 variables. The first variable is the class label (1, 2, 3).

Source

https://archive.ics.uci.edu/ml/datasets/wine

zoo *Zoo dataset*

Description

Animal description based on various features.

Usage

```
zoo
```

Format

The dataset has 101 instances described by 17 qualitative variables.

Source

https://archive.ics.uci.edu/ml/datasets/zoo

Index