

# Package ‘fdm2id’

October 29, 2021

**Title** Data Mining and R Programming for Beginners

**Version** 0.9.6

**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, mclust, nnet, pls

**License** GPL-3

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 7.1.1

**Imports** car, caret, class, cluster, e1071, FactoMineR, flexclust, fpc, glmnet, graphics, grDevices, ibr, irr, kohonen, leaps, MASS, mda, meanShiftR, methods, questionr, randomForest, ROCR, rpart, Rtsne, SnowballC, stats, text2vec, stopwords, utils, wordcloud, xgboost

**Suggests** datasets, fds

**Enhances** NMF

**NeedsCompilation** no

**Author** Alexandre Blansché [aut, cre]

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

**Repository** CRAN

**Date/Publication** 2021-10-29 15:00:02 UTC

## R topics documented:

accident2014 . . . . .	5
ADABOOST . . . . .	6
alcohol . . . . .	7

APRIORI	7
apriori-class	8
autopmg	9
BAGGING	9
beetles	10
birth	11
boosting-class	11
boxclus	12
britpop	12
CA	13
CART	14
cartdepth	15
cartinfo	15
cartleafs	16
cartnodes	17
cartplot	17
CDA	19
cda-class	19
closegraphics	20
compare	21
compare.accuracy	21
compare.jaccard	22
compare.kappa	23
confusion	24
cookies	25
cookplot	26
correlated	26
cost.curves	27
credit	28
data.diag	28
data.gauss	29
data.parabol	30
data.target1	31
data.target2	32
data.twomoons	33
data.xor	34
data1	35
data2	35
data3	36
dataset-class	36
dbs-class	37
DBSCAN	37
decathlon	38
distplot	39
EM	39
em-class	40
eucalyptus	41
evaluation	41

evaluation.accuracy . . . . .	42
evaluation.fmeasure . . . . .	43
evaluation.fowlkesmallows . . . . .	44
evaluation.goodness . . . . .	45
evaluation.jaccard . . . . .	46
evaluation.kappa . . . . .	47
evaluation.msep . . . . .	47
evaluation.precision . . . . .	48
evaluation.r2 . . . . .	49
evaluation.recall . . . . .	50
exportgraphics . . . . .	51
exportgraphics.off . . . . .	51
factorial-class . . . . .	52
FEATURESELECTION . . . . .	53
filter.rules . . . . .	54
frequentwords . . . . .	55
general.rules . . . . .	56
getvocab . . . . .	57
GRADIENTBOOSTING . . . . .	58
HCA . . . . .	59
intern . . . . .	59
intern.dunn . . . . .	60
intern.interclass . . . . .	61
intern.intraclass . . . . .	62
ionosphere . . . . .	62
keiser . . . . .	63
KERREG . . . . .	64
KMEANS . . . . .	64
kmeans.getk . . . . .	65
KNN . . . . .	66
knn-class . . . . .	67
LDA . . . . .	68
leverageplot . . . . .	68
LINREG . . . . .	69
linsep . . . . .	70
loadtext . . . . .	71
LR . . . . .	72
MCA . . . . .	73
MEANSHIFT . . . . .	74
meanshift-class . . . . .	75
MLP . . . . .	75
MLPREG . . . . .	77
model-class . . . . .	78
movies . . . . .	78
NB . . . . .	79
NMF . . . . .	79
ozone . . . . .	80
params-class . . . . .	81

PCA	81
performance	82
plot.cda	84
plot.factorial	85
plot.som	86
plotcloud	86
plotclus	87
plotdata	88
plotzipf	89
POLYREG	90
predict.apriori	91
predict.boosting	92
predict.cda	93
predict.dbs	93
predict.em	94
predict.kmeans	95
predict.knn	95
predict.meanshift	96
predict.model	97
predict.selection	98
predict.textmining	99
print.apriori	100
print.factorial	100
pseudoF	101
QDA	102
query.docs	102
query.words	103
RANDOMFOREST	104
reg1	105
reg2	106
regplot	106
resplot	107
roc.curves	107
rotation	108
runningtime	109
scatterplot	109
selectfeatures	110
selection-class	111
snore	112
SOM	113
som-class	114
SPECTRAL	114
spectral-class	115
spine	115
splitdata	116
stability	117
STUMP	118
summary.apriori	119

SVD . . . . .	119
SVM . . . . .	120
SVMl . . . . .	121
SVMr . . . . .	122
SVR . . . . .	123
SVRl . . . . .	124
SVRr . . . . .	126
temperature . . . . .	127
TEXTMINING . . . . .	127
textmining-class . . . . .	128
titanic . . . . .	129
treeplot . . . . .	129
TSNE . . . . .	130
universite . . . . .	131
vectorize.docs . . . . .	131
vectorize.words . . . . .	133
vectorizer-class . . . . .	134
vowels . . . . .	135
wheat . . . . .	135
wine . . . . .	136
zoo . . . . .	136
<b>Index</b>	<b>137</b>

---

accident2014

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


---

## Description

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

## Usage

accident2014

## Format

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

## Source

<https://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	<i>Alcohol dataset</i>
---------	------------------------

---

**Description**

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

**Usage**

```
alcohol
```

**Format**

The dataset has 27 instances described by 4 variables. The variables are the average amount of alcohol of different types per year per inhabitant.

**Source**

<https://www.who.int/>

---

APRIORI	<i>Classification using APRIORI</i>
---------	-------------------------------------

---

**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 <code>data.frame</code> .
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 parameters 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](#)



---

autompg	<i>Auto MPG dataset</i>
---------	-------------------------

---

**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	<i>Classification using Bagging</i>
---------	-------------------------------------

---

**Description**

Ensemble learning, through Bagging Algorithm.

**Usage**

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

**Arguments**

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

---

**Description**

Tutorial data set (vector).

**Usage**

birth

**Format**

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

---

boosting-class	<i>Boosting methods model</i>
----------------	-------------------------------

---

**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	<i>Clustering Box Plots</i>
---------	-----------------------------

---

**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	<i>Population and location of 18 major british cities.</i>
---------	--

---

**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**

<code>d</code>	A ddata frame or a table with n rows and p columns, i.e. a contingency table.
<code>ncp</code>	The number of dimensions kept in the results (by default 5).
<code>row.sup</code>	A vector indicating the indexes of the supplementary rows.
<code>col.sup</code>	A vector indicating the indexes of the supplementary columns.
<code>quanti.sup</code>	A vector indicating the indexes of the supplementary continuous variables.
<code>quali.sup</code>	A vector indicating the indexes of the categorical supplementary variables.
<code>row.w</code>	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

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

## Value

The classification model.

## See Also

[cartdepth](#), [cartinfo](#), [cartleaves](#), [cartnodes](#), [cartplot](#), [rpart](#)

## Examples

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

---

cartdepth	<i>Depth</i>
-----------	--------------

---

**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	<i>CART information</i>
----------	-------------------------

---

**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](#), [cartleaves](#), [cartnodes](#), [cartplot](#)

**Examples**

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

---

cartleaves	<i>Number of Leafs</i>
------------	------------------------

---

**Description**

Return the number of leaves of a decision tree.

**Usage**

```
cartleaves(model)
```

**Arguments**

model            The decision tree.

**Value**

The number of leaves.

**See Also**

[CART](#), [cartdepth](#), [cartinfo](#), [cartnodes](#), [cartplot](#)

**Examples**

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



---

cartnodes	<i>Number of Nodes</i>
-----------	------------------------

---

**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](#), [cartleaves](#), [cartplot](#)

**Examples**

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

---

cartplot	<i>CART Plot</i>
----------	------------------

---

**Description**

Plot a decision tree obtained by CART.

**Usage**

```
cartplot(
  model,
  margin = 0.2,
  branch = 0.3,
  uniform = TRUE,
  fancy = TRUE,
  pretty = TRUE,
```

```

    fwidth = 0,
    fheight = 0,
    ...
)

```

### Arguments

model	The decision tree.
margin	an extra fraction of white space to leave around the borders of the tree. (Long labels sometimes get cut off by the default computation).
branch	controls the shape of the branches from parent to child node. Any number from 0 to 1 is allowed. A value of 1 gives square shouldered branches, a value of 0 give V shaped branches, with other values being intermediate.
uniform	if TRUE, uniform vertical spacing of the nodes is used; this may be less cluttered when fitting a large plot onto a page. The default is to use a non-uniform spacing proportional to the error in the fit.
fancy	Logical. If TRUE, nodes are represented by ellipses (interior nodes) and rectangles (leaves) and labeled by yval. The edges connecting the nodes are labeled by left and right splits.
pretty	an alternative to the minlength argument, see <a href="#">labels.rpart</a> .
fwidth	Relates to option fancy and the width of the ellipses and rectangles. If fwidth < 1 then it is a scaling factor (default = 0.8). If fwidth > 1 then it represents the number of character widths (for current graphical device) to use.
fheight	Relates to option fancy and the width of the ellipses and rectangles. If fwidth < 1 then it is a scaling factor (default = 0.8). If fwidth > 1 then it represents the number of character heights (for current graphical device) to use.
...	Other parameters.

### See Also

[CART](#), [cartdepth](#), [cartinfo](#), [cartleafs](#), [cartnodes](#)

### Examples

```

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

```

**Description**

This function builds a classification model using Canonical Discriminant Analysis.

**Usage**

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

**Arguments**

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

**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	<i>Comparison of two sets of clusters</i>
---------	---

---

**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 evaluation 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$cluster, iris [, 5])
compare (km$cluster, iris [, 5], eval = c ("accuracy", "kappa"), comp = "pairwise")
```

---

compare.accuracy	<i>Comparison of two sets of clusters, using accuracy</i>
------------------	---

---

**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	<i>Comparison of two sets of clusters, using kappa</i>
---------------	--

---

**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	<i>Confuion matrix</i>
-----------	------------------------

---

**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	<i>Plot the Cook's distance of a linear regression model</i>
----------	--

---

**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	<i>Correlated variables</i>
------------	-----------------------------

---

**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	<i>Plot Cost Curves</i>
-------------	-------------------------

---

**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	<i>Credit dataset</i>
--------	-----------------------

---

**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	<i>Square dataset</i>
-----------	-----------------------

---

**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	<i>Gaussian mixture dataset</i>
------------	---------------------------------

---

**Description**

Generate a random multidimensional 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	<i>Parabol dataset</i>
--------------	------------------------

---

**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	<i>Target1 dataset</i>
--------------	------------------------

---

**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	<i>Target2 dataset</i>
--------------	------------------------

---

### 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	<i>Two moons dataset</i>
---------------	--------------------------

---

### 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	<i>XOR dataset</i>
----------	--------------------

---

### 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 ( $2^{\text{ndim}}$ 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	<i>"data1" dataset</i>
-------	------------------------

---

**Description**

Synthetic dataset.

**Usage**

data1

**Format**

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

**Author(s)**

Alexandre Blansch  <alexandre.blansche@univ-lorraine.fr>

---

data2	<i>"data2" dataset</i>
-------	------------------------

---

**Description**

Synthetic dataset.

**Usage**

data2

**Format**

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

**Author(s)**

Alexandre Blansch  <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	<i>DBSCAN model</i>
-----------	---------------------

---

**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	<i>DBSCAN clustering method</i>
--------	---------------------------------

---

**Description**

Run the DBSCAN algorithm for clustering.

**Usage**

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

**Arguments**

<code>d</code>	The dataset ( <code>matrix</code> or <code>data.frame</code> ).
<code>minpts</code>	Reachability minimum no. of points.
<code>eps</code>	Reachability distance.
<code>...</code>	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, eps = 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	<i>Plot a k-distance graphic</i>
----------	----------------------------------

---

**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 ( <i>matrix</i> or <i>data.frame</i> ).
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	<i>Expectation-Maximization clustering method</i>
----	---

---

**Description**

Run the EM algorithm for clustering.

**Usage**

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

**Arguments**

d	The dataset ( <i>matrix</i> or <i>data.frame</i> ).
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 <a href="#">mclustModelNames</a> 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  $i$ th observation belonging to the  $k$ th 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	<i>Eucalyptus dataset</i>
------------	---------------------------

---

**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/~lepenec/fr/teaching/>

---

evaluation	<i>Evaluation of classification or regression predictions</i>
------------	---

---

**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, targets, ...)
```

**Arguments**

predictions	The predictions of a classification model (factor or vector).
targets	Actual targets of the dataset (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.fmeasure     *F-measure*

---

**Description**

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

**Usage**

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

**Arguments**

predictions	The predictions of a classification model (factor or vector).
targets	Actual targets of the dataset (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,
  targets,
  positive = levels(targets)[1],
  ...
)
```

## Arguments

predictions	The predictions of a classification model (factor or vector).
targets	Actual targets of the dataset (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	<i>Goodness</i>
---------------------	-----------------

---

## Description

Evaluation predictions of a classification model according to Goodness index.

## Usage

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

## Arguments

predictions	The predictions of a classification model (factor or vector).
targets	Actual targets of the dataset (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, targets, positive = levels(targets)[1], ...)
```

## Arguments

predictions	The predictions of a classification model (factor or vector).
targets	Actual targets of the dataset (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	<i>Kappa evaluation of classification predictions</i>
------------------	---

---

**Description**

Evaluation predictions of a classification model according to kappa.

**Usage**

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

**Arguments**

predictions	The predictions of a classification model (factor or vector).
targets	Actual targets of the dataset (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	<i>MSEP evaluation of regression predictions</i>
-----------------	--

---

**Description**

Evaluation predictions of a regression model according to MSEP

**Usage**

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

**Arguments**

predictions     The predictions of a regression model (vector).  
 targets         Actual targets of the dataset (vector).

**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, targets, positive = levels(targets)[1], ...)
```

**Arguments**

predictions     The predictions of a classification model (factor or vector).  
 targets         Actual targets of the dataset (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, targets)
```

**Arguments**

predictions      The predictions of a regression model (vector).  
targets            Actual targets of the dataset (vector).

**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	<i>Recall of classification predictions</i>
-------------------	---

---

**Description**

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

**Usage**

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

**Arguments**

predictions	The predictions of a classification model (factor or vector).
targets	Actual targets of the dataset (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	<i>Open a graphics device</i>
----------------	-------------------------------

---

**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	<i>Toggle graphic exports</i>
--------------------	-------------------------------

---

**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

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

---

**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 <code>getvocab</code> ).
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	<i>Remove redundancy in a set of rules</i>
---------------	--

---

**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	<i>Extract words and phrases from a corpus</i>
----------	--

---

**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**

<code>train</code>	The training set (description), as a <code>data.frame</code> .
<code>labels</code>	Class labels of the training set (vector or factor).
<code>ntree</code>	The number of trees in the forest.
<code>learningrate</code>	The learning rate (between 0 and 1).
<code>tune</code>	If true, the function returns parameters instead of a classification model.
<code>...</code>	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	<i>Clustering evaluation through intraclass inertia</i>
-------------------	---

---

### 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	<i>Ionosphere dataset</i>
------------	---------------------------

---

### 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 database 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>

---

keiser

*Keiser rule*

---

**Description**

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

**Usage**

```
keiser(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)
keiser (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 parameters 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 number 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 <a href="#">kmeans</a> 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 parameters 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**

<code>train</code>	The training set (description), as a <code>data.frame</code> .
<code>labels</code>	Class labels of the training set (vector or factor).
<code>tune</code>	If true, the function returns parameters instead of a classification model.
<code>...</code>	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,
  formula = ".",
  reg = c("linear", "subset", "ridge", "lasso", "elastic", "pcr", "plsr"),
  regeval = c("r2", "bic", "adjr2", "cp", "mse"),
  scale = TRUE,
  lambda = 10^seq(-5, 5, length.out = 101),
  alpha = 0.5,
  graph = TRUE,
  tune = FALSE,
  ...
)
```

**Arguments**

x	Predictor matrix.
y	Response vector.
formula	A symbolic description of the model to be fitted (as a character string).
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], formula = "-1+supp+dose") # Different intersept
LINREG (ToothGrowth [, -1], ToothGrowth [, 1], formula = "dose:supp") # Different slope
LINREG (ToothGrowth [, -1], ToothGrowth [, 1], formula = "-1+supp+dose:supp") # 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 = "pls")

## 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	<i>load a text file</i>
----------	-------------------------

---

**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

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

<code>d</code>	A ddata frame or a table with n rows and p columns, i.e. a contingency table.
<code>ncp</code>	The number of dimensions kept in the results (by default 5).
<code>ind.sup</code>	A vector indicating the indexes of the supplementary individuals.
<code>quanti.sup</code>	A vector indicating the indexes of the quantitative supplementary variables.
<code>quali.sup</code>	A vector indicating the indexes of the categorical supplementary variables.
<code>row.w</code>	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,  
  kernel = "NORMAL",  
  bandwidth = rep(1, ncol(d)),  
  alpha = 0,  
  iterations = 10,  
  epsilon = 1e-08,  
  epsilonCluster = 1e-04,  
  ...  
)
```

**Arguments**

d	The dataset (matrix or data.frame).
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.
...	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	<i>MeanShift model</i>
-----------------	------------------------

---

### 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	<i>Classification using Multilayer Perceptron</i>
-----	---

---

### 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**

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

---

**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	<i>Movies dataset</i>
--------	-----------------------

---

**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**

<code>train</code>	The training set (description), as a <code>data.frame</code> .
<code>labels</code>	Class labels of the training set (vector or factor).
<code>tune</code>	If true, the function returns parameters instead of a classification model.
<code>...</code>	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 contains 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(quant.sup) - length(quali.sup),  
  ind.sup = NULL,  
  quanti.sup = NULL,  
  quali.sup = NULL,  
  row.w = NULL,  
  col.w = NULL  
)
```

**Arguments**

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

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

## 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	<i>Plot function for factorial-class</i>
----------------	--

---

**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**

<code>x</code>	The Kohonen's map (object of class <a href="#">som-class</a> ).
<code>type</code>	The type of plot.
<code>col</code>	Color of the data points
<code>labels</code>	A vector of character strings to be printed instead of points in the plot.
<code>...</code>	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
```

---

`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 <code>getvocab</code> ).
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 shown on the (scatter) plot.
...	Other parameters.

**Examples**

```

require (datasets)
data (iris)
# Without classification
plotdata (iris [, -5]) # Défaut (pairs)
# With classification
plotdata (iris [, -5], iris [, 5]) # Défaut (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]) # Défaut (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]) # Défaut (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

```

**Description**

Plot the frequency of words in a document against the ranks of those words. It also plots 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 <code>getvocab</code> ).
--------	--

**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 polynomial degree.
tune	If true, the function returns parameters 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	<i>Model predictions</i>
-----------------	--------------------------

---

**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 <code>apriori</code> , created by <code>apriori.classif</code> ).
test	The test set (a <code>data.frame</code> ).
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 <a href="#">boosting-class</a> , created by <a href="#">ADABOOST</a> or <a href="#">BAGGING</a> ).
test	The test set (a <code>data.frame</code> )
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	<i>Model predictions</i>
-------------	--------------------------

---

**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 <a href="#">cda-class</a> , created by <a href="#">CDA</a> ).
test	The test set (a <code>data.frame</code> )
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	<i>Predict function for DBSCAN</i>
-------------	------------------------------------

---

**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 <a href="#">dbs-class</a> , created by <a href="#">DBSCAN</a> ).
newdata	A new dataset (a <code>data.frame</code> ), 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 <a href="#">em-class</a> , created by <a href="#">EM</a> ).
newdata	A new dataset (a <code>data.frame</code> ), 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	<i>Predict function for K-means</i>
----------------	-------------------------------------

---

**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 <a href="#">KMEANS</a> ).
newdata	A new dataset (a <code>data.frame</code> ), 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	<i>Model predictions</i>
-------------	--------------------------

---

**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 <code>knn</code> ).
test	The test set (a <code>data.frame</code> ).
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 <code>MEANSHIFT</code> ).
newdata	A new dataset (a <code>data.frame</code> ), 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	<i>Model predictions</i>
---------------	--------------------------

---

## 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 <a href="#">cda-class</a> , created by <a href="#">CDA</a> ).
test	The test set (a <code>data.frame</code> ).
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 <a href="#">cda-class</a> , created by <a href="#">CDA</a> ).
test	The test set (a <code>data.frame</code> ).
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 <a href="#">textmining-class</a> , created by <a href="#">TEXTMINING</a> ).
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

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

<code>train</code>	The training set (description), as a <code>data.frame</code> .
<code>labels</code>	Class labels of the training set (vector or factor).
<code>ntree</code>	The number of trees in the forest.
<code>nvar</code>	Number of variables randomly sampled as candidates at each split.
<code>tune</code>	If true, the function returns paramters instead of a classification model.
<code>...</code>	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	<i>reg1 dataset</i>
------	---------------------

---

**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 regression 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	<i>Plot the studentized residuals of a linear regression model</i>
---------	--

---

**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	<i>Plot ROC Curves</i>
------------	------------------------

---

**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	<i>Running time</i>
-------------	---------------------

---

**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	<i>Clustering Scatter Plots</i>
-------------	---------------------------------

---

**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	<i>Feature selection for classification</i>
----------------	---

---

**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	<i>Feature selection</i>
-----------------	--------------------------

---

**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>



**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**

<code>d</code>	The dataset (matrix or data.frame).
<code>xdim, ydim</code>	The dimensions of the grid.
<code>rlen</code>	The number of iterations.
<code>post</code>	The post-treatment method: "none" (None), "single" (Single link) or "ward" (Ward clustering).
<code>k</code>	The number of cluster (only used if post is different from "none").
<code>...</code>	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	<i>Spectral clustering model</i>
----------------	----------------------------------

---

**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	<i>Spine dataset</i>
-------	----------------------

---

**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**

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

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

**Value**

The classification model.

**See Also**

[CART](#)

**Examples**

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

---

summary.apriori	<i>Print summary of a classification model obtained by APRIORI</i>
-----------------	--

---

**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	<i>Singular Value Decomposition</i>
-----	-------------------------------------

---

**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](#), [SVM1](#), [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)
```

---

SVM1

---

*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**

```
SVM1(
  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 parameters instead of a classification model.  
 ...                     Other arguments.

**Value**

The classification model.

**See Also**

[svm](#), [SVM](#)

**Examples**

```
## Not run:
require (datasets)
data (iris)
SVM1 (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 choose 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**

```
SVRI(  
  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)  
SVRI (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	<i>Temperature dataset</i>
-------------	----------------------------

---

**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	<i>Text mining</i>
------------	--------------------

---

**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	<i>Titanic dataset</i>
---------	------------------------

---

**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](#)

---

treepplot	<i>Dendrogram Plots</i>
-----------	-------------------------

---

**Description**

Draws a dendrogram.

**Usage**

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

**Arguments**

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

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

**See Also**

[Rtsne](#)

**Examples**

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

---

universite	<i>University dataset</i>
------------	---------------------------

---

**Description**

The dataset presents a french university demographics.

**Usage**

```
universite
```

**Format**

The dataset has 10 instances (university departments) described by 12 variables. The first 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	<i>Document vectorization</i>
----------------	-------------------------------

---

**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	<i>Word vectorization</i>
-----------------	---------------------------

---

### 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-occurrence 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	<i>Document vectorization object</i>
------------------	--------------------------------------

---

**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	<i>Vowels dataset</i>
--------	-----------------------

---

**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	<i>Wheat dataset</i>
-------	----------------------

---

**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 variety.

**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

accident2014, 5  
ADABOOST, 6, 10, 11, 92  
agnes, 59, 130  
alcohol, 7  
APRIORI, 7, 8, 91, 100, 119  
apriori, 8, 54, 56, 91, 100, 119  
apriori-class, 8  
autompg, 9

BAGGING, 6, 9, 11, 92  
beetles, 10  
birth, 11  
boosting-class, 11  
boxclus, 12, 88  
boxplot, 12  
britpop, 12

CA, 13, 13, 52, 73, 82, 85, 101  
CART, 14, 15–18, 118  
cartdepth, 14, 15, 16–18  
cartinfo, 14, 15, 15, 16–18  
cartleaves, 14–16, 16, 17, 18  
cartnodes, 14–16, 17, 18  
cartplot, 14–17, 17  
CDA, 19, 20, 84, 93, 97, 98  
cda-class, 19  
closegraphics, 20, 51, 52  
compare, 21, 22, 23, 60, 117  
compare.accuracy, 21, 21, 22, 23  
compare.jaccard, 21, 22, 22, 23  
compare.kappa, 21, 22, 23  
confusion, 24, 42, 83  
cookies, 25  
cookplot, 26  
cor, 27  
correlated, 26  
cost.curves, 27, 83, 108  
create\_vocabulary, 57  
credit, 28

data.diag, 28, 30–34  
data.gauss, 29, 34  
data.parabol, 29, 30, 30, 31–34  
data.target1, 29, 30, 31, 32, 33  
data.target2, 29–31, 32, 33, 34  
data.twomoons, 29–32, 33, 34  
data.xor, 29–33, 34  
data1, 35  
data2, 35  
data3, 36  
dataset-class, 36  
dbs-class, 37  
DBSCAN, 37, 37, 39, 94  
dbscan, 37, 39  
decathlon, 38  
dendrogram, 130  
dev.off, 20  
Devices, 51  
difftime, 109  
distplot, 37, 39  
download.file, 71

EM, 39, 40, 94  
em, 40  
em-class, 40  
eucalyptus, 41  
evaluation, 24, 41, 43–50, 83  
evaluation.accuracy, 42, 42, 43–48, 50  
evaluation.fmeasure, 42, 43, 43, 44–48, 50  
evaluation.fowlkesmallows, 42, 43, 44, 45–48, 50  
evaluation.goodness, 42–44, 45, 46–48, 50  
evaluation.jaccard, 42–45, 46, 47, 48, 50  
evaluation.kappa, 42–47, 47, 48, 50  
evaluation.msep, 42, 47, 49  
evaluation.precision, 42–47, 48, 50  
evaluation.r2, 42, 48, 49  
evaluation.recall, 42–48, 50  
exportgraphics, 20, 51, 52  
exportgraphics.off, 51

- exportgraphics.on (exportgraphics.off),  
51
- factorial-class, 52
- FEATURESELECTION, 53, 98, 111, 112
- filter.rules, 54
- frequentwords, 55
- general.rules, 56
- getvocab, 55, 57, 87, 90
- glmnet, 70
- GRADIENTBOOSTING, 58
- HCA, 59, 130
- hclust, 130
- intern, 21, 59, 61, 62, 117
- intern.dunn, 60, 60, 61, 62
- intern.interclass, 60, 61, 61, 62
- intern.intraclass, 60, 61, 62
- ionosphere, 62
- keiser, 63, 82
- KERREG, 64
- KMEANS, 64, 95, 101
- kmeans, 65, 66, 101
- kmeans.getk, 65, 101
- KNN, 66, 67, 95, 96
- knn, 67, 96
- knn-class, 67
- labc, 25
- labels.rpart, 18
- labp, 25
- LDA, 68
- lda, 68
- leverageplot, 68
- LINREG, 69
- linsep, 70
- lm, 70
- loadtext, 71
- LR, 72
- MCA, 13, 52, 73, 73, 82, 85, 101
- mclustModelNames, 39, 40
- MEANSHIFT, 74, 75, 96
- meanShift, 74
- meanshift-class, 75
- MLP, 75, 81
- MLPREG, 77, 81
- model-class, 78
- movies, 78
- mstep, 40
- multinom, 72
- mvr, 70
- naiveBayes, 79
- NB, 79
- nirc, 25
- nirp, 25
- NMF, 79
- nmf, 80
- nnet, 76, 77
- npregress, 64
- ozone, 80
- params-class, 81
- PCA, 13, 52, 63, 73, 81, 82, 85, 101
- performance, 24, 27, 42, 82, 108
- plot.CA, 85
- plot.cda, 19, 20, 84, 93
- plot.factorial, 13, 52, 73, 82, 85
- plot.MCA, 85
- plot.PCA, 85
- plot.som, 86, 88, 113, 114
- plotcloud, 86, 90
- plotclus, 87
- plotdata, 88
- plotzipf, 57, 87, 89
- POLYREG, 90
- polyreg, 91
- predict, 78
- predict.apriori, 8, 91, 100, 119
- predict.boosting, 6, 10, 11, 92
- predict.cda, 19, 20, 84, 93
- predict.dbs, 37, 93
- predict.em, 94
- predict.kmeans, 65, 95
- predict.knn, 67, 95
- predict.meanshift, 74, 96
- predict.model, 78, 97
- predict.selection, 53, 98, 112
- predict.textmining, 99, 128
- print.apriori, 8, 100, 119
- print.CA, 101
- print.factorial, 100
- print.MCA, 101
- print.PCA, 101

- pseudof, [66](#), [101](#)
- QDA, [102](#)
- qda, [102](#)
- query.docs, [102](#), [132](#), [134](#)
- query.words, [103](#), [134](#)
- RANDOMFOREST, [104](#)
- randomForest, [105](#)
- reg1, [105](#)
- reg2, [106](#)
- regplot, [106](#)
- regsubsets, [70](#)
- resplot, [107](#)
- roc.curves, [27](#), [83](#), [107](#)
- rotation, [108](#)
- rpart, [14](#)
- Rtsne, [130](#)
- runningtime, [109](#)
- scatterplot, [88](#), [109](#)
- selectfeatures, [53](#), [110](#), [112](#)
- selection-class, [111](#)
- sim2, [103](#), [104](#)
- snore, [112](#)
- SOM, [86](#), [113](#), [114](#)
- som, [113](#), [114](#)
- som-class, [114](#)
- SPECTRAL, [114](#), [115](#)
- spectral-class, [115](#)
- spine, [115](#)
- splitdata, [24](#), [36](#), [116](#)
- stability, [21](#), [60](#), [117](#)
- stopwords, [57](#), [132](#), [134](#)
- STUMP, [118](#)
- subset, [54](#)
- summary.apriori, [8](#), [100](#), [119](#)
- SVD, [119](#)
- svd, [120](#)
- SVM, [81](#), [120](#), [122](#), [123](#)
- svm, [121–126](#)
- SVM1, [121](#), [121](#)
- SVMr, [121](#), [122](#)
- SVR, [81](#), [123](#), [125](#), [126](#)
- SVR1, [124](#), [124](#)
- SVRr, [124](#), [126](#)
- temperature, [127](#)
- TEXTMINING, [99](#), [127](#), [128](#)
- textmining-class, [128](#)
- Titanic, [129](#)
- titanic, [129](#)
- toggleexport, [20](#), [51](#)
- toggleexport (exportgraphics.off), [51](#)
- treemap, [88](#), [129](#)
- TSNE, [130](#)
- universite, [131](#)
- unzip, [71](#)
- vectorize.docs, [103](#), [128](#), [131](#), [134](#)
- vectorize.words, [104](#), [128](#), [133](#)
- vectorizer-class, [134](#)
- vectorizers, [132](#), [134](#)
- vowels, [135](#)
- wheat, [135](#)
- wine, [136](#)
- wordcloud, [87](#)
- xgboost, [58](#)
- zoo, [136](#)