Package 'fdm2id'

September 4, 2026

Title Data Mining and R Programming for Beginners

Version 1.0.1

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

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

Imports graphics, grDevices, Matrix, mclust, methods, pls, stats, utils

Suggests car, caret, class, cluster, datasets, e1071, fds, flexclust, fpc, glmnet, ibr, irr, knitr, kohonen, leaps, MASS, mda, meanShiftR, mlbench, questionr, randomForest, rmarkdown, RSpectra, ROCR, rpart, rpart.plot, Rtsne, SnowballC, stopwords, testthat (>= 3.0.0), text2vec, wordcloud, xgboost (>= 2.1.0)

Enhances NMF

License GPL-3

Encoding UTF-8

LazyData true

Config/roxygen2/version 8.1.0

Config/testthat/edition 3

VignetteBuilder knitr

NeedsCompilation no

Author Alexandre Blansché [aut, cre]

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

Repository CRAN

Date/Publication 2026-09-04 15:50:02 UTC

Contents

accident2014 . . . 6
ADABOOST . . . 6
alcohol . . . 7
APRIORI . . . 8
apriori-class . . . 9
augmentation . . . 10
autompg . . . 10
BAGGING . . . 11
beetles . . . 12
birth . . . 13
boosting-class . . . 13
boxclus . . . 14
britpop . . . 14
CA . . . 15
capitals . . . 16
CART . . . 16
cartdepth . . . 18
cartinfo . . . 18
cartleafs . . . 19
cartnodes . . . 20
cartplot . . . 20
CDA . . . 21
cda-class . . . 22
closegraphics . . . 23
compare . . . 23
compare.accuracy . . . 24
compare.jaccard . . . 25
compare.kappa . . . 26
confusion . . . 27
cookies . . . 28
cookplot . . . 29
correlated . . . 30
cost.curves . . . 30
credit . . . 31
data.diag . . . 32
data.gauss . . . 33
data.parabol . . . 34
data.target1 . . . 35
data.target2 . . . 36
data.twomoons . . . 37
data.xor . . . 38
data1 . . . 39
data2 . . . 39
data3 . . . 40
dataset-class . . . 40
dbs-class . . . 41

DBSCAN . . . 41
decathlon . . . 42
distplot . . . 43
EM . . . 43
em-class . . . 44
eucalyptus . . . 45
evaluation . . . 45
evaluation.accuracy . . . 46
evaluation.adjr2 . . . 47
evaluation.fmeasure . . . 48
evaluation.fowlkesmallows . . . 49
evaluation.goodness . . . 51
evaluation.jaccard . . . 52
evaluation.kappa . . . 54
evaluation.msep . . . 55
evaluation.precision . . . 55
evaluation.r2 . . . 57
evaluation.recall . . . 57
exportgraphics . . . 59
exportgraphics.off . . . 60
factorial-class . . . 61
FEATURESELECTION . . . 61
filter.rules . . . 63
frequentwords . . . 64
GBREG . . . 65
general.rules . . . 66
getvocab . . . 67
GRADIENTBOOSTING . . . 68
HCA . . . 69
intern . . . 70
intern.dunn . . . 71
intern.interclass . . . 72
intern.intraclass . . . 72
ionosphere . . . 73
kaiser . . . 74
KERREG . . . 74
KMEANS . . . 75
kmeans.getk . . . 77
KNN . . . 78
knn-class . . . 79
LDA . . . 80
leverageplot . . . 81
LINREG . . . 81
linsep . . . 83
loadtext . . . 84
LR . . . 85
MCA . . . 87
MEANSHIFT . . . 88

meanshift-class . . . 89
MLP . . . 90
MLPREG . . . 91
model-class . . . 92
movies . . . 93
NB . . . 93
NMF . . . 94
ozone . . . 95
PAM . . . 96
params-class . . . 97
PCA . . . 97
performance . . . 99
plot.apriori . . . 101
plot.cda . . . 102
plot.factorial . . . 103
plot.selection . . . 104
plot.som . . . 105
plotavsp . . . 106
plotcloud . . . 107
plotclus . . . 107
plotdata . . . 109
plotzipf . . . 111
POLYREG . . . 112
predict.apriori . . . 113
predict.boosting . . . 114
predict.cda . . . 115
predict.dbs . . . 115
predict.em . . . 116
predict.factorial . . . 117
predict.hca . . . 118
predict.kmeans . . . 119
predict.knn . . . 119
predict.meanshift . . . 120
predict.model . . . 121
predict.pam . . . 122
predict.selection . . . 122
predict.som . . . 123
predict.spectral . . . 124
predict.textmining . . . 125
print.apriori . . . 126
print.boosting . . . 126
print.cda . . . 127
print.dataset . . . 128
print.dbs . . . 128
print.em . . . 129
print.factorial . . . 130
print.knn . . . 130
print.meanshift . . . 131

print.model . . . 132
print.params . . . 132
print.selection . . . 133
print.som . . . 134
print.spectral . . . 134
pseudoF . . . 135
QDA . . . 136
query.docs . . . 137
query.words . . . 138
RANDOMFOREST . . . 139
reg1 . . . 140
reg2 . . . 140
regplot . . . 141
resplot . . . 141
roc.curves . . . 142
rotation . . . 143
runningtime . . . 144
scatterplot . . . 145
selectfeatures . . . 146
selection-class . . . 147
snore . . . 148
SOM . . . 148
som-class . . . 149
SPECTRAL . . . 150
spectral-class . . . 151
spine . . . 151
splitdata . . . 152
stability . . . 153
STUMP . . . 154
summary.apriori . . . 155
summary.model . . . 156
SVD . . . 156
SVM . . . 157
SVMl . . . 158
SVMr . . . 160
SVR . . . 161
SVRl . . . 163
SVRr . . . 164
temperature . . . 165
TEXTMINING . . . 166
textmining-class . . . 167
titanic . . . 167
treeplot . . . 168
TSNE . . . 169
universite . . . 170
vectorize.docs . . . 170
vectorize.words . . . 172
vectorizer-class . . . 173

vowels 174
wheat 174
wine 175
zoo 175

Index **176**

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

Description

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

Usage

```
accident2014
```

Format

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

Source

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

`ADABOOST` *Classification using AdaBoost*

Description

Ensemble learning, through AdaBoost Algorithm.

Usage

```
ADABOOST(
  x,
  y,
  learningmethod,
  nsamples = 100,
  fuzzy = FALSE,
  tune = FALSE,
  methodparameters = NULL,
  graph = 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 parameters instead of a classification model.
methodparameters	Present for interface consistency with performance (which always passes it when fitting a model). Currently unused: ADABOOST does not support reusing pre-tuned parameters (the base learner given as learningmethod is tuned independently on each boosting sample).
graph	Present for interface consistency with performance (which always passes it when fitting a model). Currently unused: ADABOOST does not produce a plot.
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

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

---

alcohol *Alcohol dataset*

Description

This dataset has been extracted from the WHO database and depicts alcohol consumption habits in 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 consumed per year per inhabitant.

Source

https://www.who.int/

APRIORI *Classification using APRIORI*

Description

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

Usage

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

Arguments

train	The training set (description), as a data.frame.
labels	Class labels of the training set (vector or factor).
supp	The minimal support of an item set (numeric value).
conf	The minimal confidence of an item set (numeric value).
prune	A logical indicating whether to prune redundant rules or not (default: FALSE).
tune	If true, the function returns parameters instead of a classification model.
methodparameters	Present for interface consistency with performance (which always passes it when fitting a model). Currently unused: APRIORI does not support reusing pre-tuned parameters.
graph	Present for interface consistency with performance (which always passes it when fitting a model). Currently unused: APRIORI does not produce a plot.

seed	A specified seed for random number generation, so that two runs on the same data give the same model. Every learning method accepts it, so that it can be set the same way whatever the method; the deterministic ones simply have nothing to draw and give the same model with or without it.
...	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.

Details

Objects of this class are plain lists with the following components:

rules The set of rules obtained by APRIORI.

transactions The training set as a transaction object.

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

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

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

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

See Also

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

augmentation *Duplicate and add noise to a dataset*

Description

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

Usage

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

Arguments

dataset	The dataset to be split (data.frame or matrix).
target	The column index (numeric) or column name (character) of the target variable (class label or response variable).
n	The scaling factor (as an integer value): the output contains n times the original dataset (the original rows, plus n - 1 noisy copies).
sigma	The baseline variance for the noise generation.
seed	A specified seed for random number generation.

Value

An augmented dataset.

Examples

```
require (datasets)
data (iris)
d = augmentation (iris, 5)
summary (iris)
summary (d)
# 'target' can also be given as a column name
d = augmentation (iris, "Species")
```

autompg *Auto MPG dataset*

Description

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

Usage

```
autompg
```

Format

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

Source

https://archive.ics.uci.edu/dataset/9/auto+mpg

BAGGING *Classification using Bagging*

Description

Ensemble learning, through Bagging Algorithm.

Usage

```
BAGGING(
  x,
  y,
  learningmethod,
  nsamples = 100,
  bag.size = nrow(x),
  tune = FALSE,
  methodparameters = NULL,
  graph = 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.
bag.size	The size of the samples.
tune	If true, the function returns parameters instead of a classification model.
methodparameters	Present for interface consistency with performance (which always passes it when fitting a model). Currently unused: BAGGING does not support reusing pre-tuned parameters (the base learner is fitted afresh on each sample).

graph	Present for interface consistency with performance (which always passes it when fitting a model). Currently unused: BAGGING does not produce a plot.
seed	A specified seed for random number generation.
...	Other specific parameters for the leaning method.

Value

The classification model.

See Also

ADABOOST, predict.boosting

Examples

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

---

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).

Species Species of flea beetle from the genus *Chaetocnema*.

Source

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

birth *Birth dataset*

Description

Tutorial data set (vector).

Usage

```
birth
```

Format

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

boosting-class *Boosting methods model*

Description

This class contains the ensemble of models obtained by a boosting or bagging method (ADABOOST, BAGGING).

Details

Objects of this class are plain lists with the following components:

`models` List of models.

`x` The learning set.

`y` The target values.

`nsamples` The number of models that were asked for. Boosting keeps fewer when it runs out of models better than chance; `print.boosting` says so.

See Also

ADABOOST, BAGGING, `predict.boosting`

boxclus *Clustering Box Plots*

Description

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

Usage

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

Arguments

d	The dataset (matrix or data.frame).
clusters	Cluster labels of the training set: a numeric vector (0 marking the observations a density method left as noise), or a factor/character vector, whose levels are then used to label the legend.
legendpos	Position of the legend
...	Other parameters.

See Also

boxplot

Examples

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

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

Description

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

Usage

```
britpop
```

Format

The dataset has 18 instances described by 3 variables.

CA *Correspondence Analysis (CA)*

Description

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

Usage

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

Arguments

d	A data frame or a table with n rows and p columns, i.e. a contingency table.
ncp	The number of dimensions kept in the results. All of them, by default: a table of I active rows by J active columns carries `min(I, J) - 1` of them, the first dimension of a contingency table being its margins, which hold no association.
row.sup	A vector indicating the indexes of the supplementary rows.
col.sup	A vector indicating the indexes of the supplementary columns.
quanti.sup	A vector indicating the indexes of the supplementary continuous variables.
quali.sup	A vector indicating the indexes of the categorical supplementary variables.
row.w	An optional row weights (by default, a vector of 1 for uniform row weights); the weights are given only for the active individuals.

Value

The CA on the dataset.

See Also

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

Examples

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

capitals *Capitals dataset*

Description

A small corpus of English sentences about France, Germany and their capitals. It is deliberately tiny, so that the examples of the text mining functions run in a moment; a real corpus is loaded with `loadtext`.

Usage

```
capitals
```

Format

A character `vector` of 30 sentences, lowercase and free of punctuation.

Author(s)

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

See Also

`getvocab`, `vectorize.docs`, `vectorize.words`, `loadtext`

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,
  xval = 10,
  tune = FALSE,
  methodparameters = NULL,
  graph = FALSE,
  seed = NULL,
  ...
)
```

Arguments

train	The training set (description), as a data.frame.
labels	Class labels of the training set (vector or factor).
minsplit	The minimum leaf size during the learning.
maxdepth	Set the maximum depth of any node of the final tree, with the root node counted as depth 0.
cp	The complexity parameter of the tree. Cross-validation is used to determine optimal cp if NULL.
xval	The number of cross-validation folds used to choose cp, when cp is NULL. xval = nrow(train) gives a leave-one-out cross-validation, which fits one tree per observation and costs about nrow(train) / 10 times as much.
tune	If true, the function returns parameters instead of a classification model.
methodparameters	Present for interface consistency with performance (which always passes it when fitting a model). Currently unused: CART does not support reusing pre-tuned parameters.
graph	Present for interface consistency with performance (which always passes it when fitting a model). Currently unused: CART does not produce a plot.
seed	A specified seed for random number generation, so that two runs on the same data give the same model. Every learning method accepts it, so that it can be set the same way whatever the method; the deterministic ones simply have nothing to draw and give the same model with or without it.
...	Other parameters.

Value

The classification model.

See Also

cartdepth, cartinfo, cartleafs, cartnodes, cartplot, rpart

Examples

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

cartdepth *Depth*

Description

Return the depth of a decision tree.

Usage

```
cartdepth(model)
```

Arguments

model	The decision tree.

Value

The depth.

See Also

CART, cartinfo, cartleafs, cartnodes, cartplot

Examples

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

cartinfo *CART information*

Description

Return various information on a CART model.

Usage

```
cartinfo(model)
```

Arguments

model	The decision tree.

Value

Various information organized into a vector.

See Also

CART, cartdepth, cartleafs, cartnodes, cartplot

Examples

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

`cartleafs` *Number of Leafs*

Description

Return the number of leafs of a decision tree.

Usage

```
cartleafs(model)
```

Arguments

`model`	The decision tree.

Value

The number of leafs.

See Also

CART, cartdepth, cartinfo, cartnodes, cartplot

Examples

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

cartnodes *Number of Nodes*

Description

Return the number of nodes of a decision tree.

Usage

```
cartnodes(model)
```

Arguments

model	The decision tree.

Value

The number of nodes.

See Also

CART, cartdepth, cartinfo, cartleafs, cartplot

Examples

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

cartplot *CART Plot*

Description

Plot a decision tree obtained by CART.

Usage

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

Arguments

model	The decision tree.
...	Other parameters.

See Also

CART, cartdepth, cartinfo, cartleafs, cartnodes

Examples

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

CDA *Classification using Canonical Discriminant Analysis*

Description

This function builds a classification model using Canonical Discriminant Analysis.

Usage

```
CDA(
  train,
  labels,
  tune = FALSE,
  methodparameters = NULL,
  graph = FALSE,
  seed = NULL,
  ...
)
```

Arguments

train	The training set (description), as a data.frame.
labels	Class labels of the training set (vector or factor).
tune	If true, the function returns parameters instead of a classification model.
methodparameters	Present for interface consistency with performance (which always passes it when fitting a model). Currently unused: CDA does not support reusing pre-tuned parameters.
graph	Present for interface consistency with performance (which always passes it when fitting a model). Currently unused: CDA does not produce a plot.
seed	A specified seed for random number generation, so that two runs on the same data give the same model. Every learning method accepts it, so that it can be set the same way whatever the method; the deterministic ones simply have nothing to draw and give the same model with or without it.
...	Other parameters.

Details

The projection is computed from the class sizes, as the between-class scatter requires. The predictions, on the other hand, use *equal* prior probabilities – an observation goes to the nearest class centre in the canonical space, whatever the size of that class. This is the geometric reading `plot.cda` draws, and it is where `CDA` differs from `LDA`, which weights the classes by their observed frequencies: on an imbalanced problem the two do not predict the same thing.

Value

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

See Also

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

Examples

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

---

cda-class *Canonical Disciminant Analysis model*

Description

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

Details

Objects of this class are plain lists with the following components:

`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 intra-class covariance matrix. A `matrix`.

`eig` One row per canonical axis and four columns: the `eigenvalue` (of $V^{-1}B$, i.e. the squared canonical correlation, between 0 and 1), its `percentage of variance` (share of the trace) and the `cumulative` one, and the `discriminant power` – the share of the trace of $W^{-1}B$, which is what `lda` and most other software call the proportion of trace. A `matrix`, or a named `vector` when there is a single axis.

`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(export = fdm2id.globals$export)
```

Arguments

export	If given, explicitly overrides the global export toggle set by toggleexport for this call only (TRUE closes the device, FALSE is a no-op). By default, the global toggle is used, so existing code is unaffected.

See Also

exportgraphics, toggleexport, dev.off

Examples

```
## Not run:
data (iris)
exportgraphics ("export.pdf")
plotdata (iris [, -5], iris [, 5])
closegraphics()
# Explicit override, ignoring the global toggle:
closegraphics (export = TRUE)

## End(Not run)
```

---

compare *Comparison of two sets of clusters*

Description

Comparison of two sets of clusters

Usage

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

Arguments

clus	The extracted clusters.
gt	The real clusters.
eval	The evaluation criterion.
comp	How the two partitions are compared: "max" (each cluster matched with the class it agrees with most, averaged over clusters), "cluster" (the same scores, not averaged) or "pairwise" (every pair of observations, labels ignored). In "pairwise" mode the three criteria are three classical indices built on the same pair counts: "accuracy" is the Rand index, "jaccard" the Jaccard index on pairs, and "kappa" Cohen's kappa on the fourfold table of pair agreements (which, by Warrens (2008), is also the adjusted Rand index). See compare.accuracy, compare.jaccard and compare.kappa.

Value

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

See Also

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

Examples

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

## End(Not run)
```

---

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

Description

Comparison of two sets of clusters, using accuracy

Usage

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

Arguments

clus	The extracted clusters.
gt	The real clusters.
comp	How the two partitions are compared. "max" matches each extracted cluster with the class it agrees with most and averages the per-cluster scores, weighted by cluster size; "cluster" returns those scores instead of averaging them; "pairwise" ignores the labels and looks at every pair of observations, asking whether the two partitions agree on grouping it or separating it. In "pairwise" mode this function is the **Rand index**: the proportion of pairs the two partitions agree on, counting both those they group and those they keep apart. See compare.jaccard and compare.kappa for the two other readings of the same pair counts.

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 — How the two partitions are compared. "max" matches each extracted cluster with the class it agrees with most and averages the per-cluster scores, weighted by cluster size; "cluster" returns those scores instead of averaging them; "pairwise" ignores the labels and looks at every pair of observations, asking whether the two partitions agree on grouping it or separating it. In "pairwise" mode this function is the **Rand index**: the proportion of pairs the two partitions agree on, counting both those they group and those they keep apart. See compare.jaccard and compare.kappa for the two other readings of the same pair counts.

Value

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

The pairwise index

the **Jaccard index** on pairs – the pairs both partitions group together, over the pairs at least one of them groups. Unlike the Rand index of compare.accuracy it ignores the pairs both keep apart, a cell that dominates as soon as there are many clusters: 150 singletons out of 150 observations score 0.67 with the Rand index and 0 here.

See Also

compare.accuracy, compare.kappa, compare

Examples

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

---

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

Description

Comparison of two sets of clusters, using kappa

Usage

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

Arguments

clus	The extracted clusters.
gt	The real clusters.
comp	How the two partitions are compared. "max" matches each extracted cluster with the class it agrees with most and averages the per-cluster scores, weighted by cluster size; "cluster" returns those scores instead of averaging them; "pairwise" ignores the labels and looks at every pair of observations, asking whether the two partitions agree on grouping it or separating it. In "pairwise" mode this function is the **Rand index**: the proportion of pairs the two partitions agree on, counting both those they group and those they keep apart. See compare.jaccard and compare.kappa for the two other readings of the same pair counts.

Value

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

The pairwise index

Cohen's kappa on the fourfold table of pair agreements, i.e. the Rand index of compare.accuracy corrected for the agreement expected by chance. That quantity is also the Hubert-Arabie *adjusted Rand index* – a theorem of Warrens (2008), not a substitution.

References

Warrens, M.J. (2008). On the Equivalence of Cohen's Kappa and the Hubert-Arabie Adjusted Rand Index. *Journal of Classification*, 25(2), 177-183.

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 *Confusion matrix*

Description

Plot a confusion matrix. Rows are the true labels and columns the predicted ones.

Usage

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

Arguments

predictions	The prediction. **This is the first argument**, as for every other evaluation function of the package (evaluation, evaluation.accuracy, evaluation.precision, ...): passing the ground truth first returns the transposed matrix.
gt	The ground truth.
norm	Whether or not the confusion matrix is normalized
graph	Whether or not a graphic is displayed.
...	Other parameters, ignored. performance forwards to its evaluation function the same ... it forwards to the learning method, so a call such as performance (BAGGING, ..., type = "confusion", learningmethod = CART) would otherwise fail on the unused learningmethod.

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$test.x)
confusion (pred, d$test.y)
```

---

cookies *Cookies dataset*

Description

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

Usage

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

Format

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

Source

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

See Also

labp, labc, nirp, nirc

---

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

Description

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

Usage

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

Arguments

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

Examples

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

correlated *Correlated variables*

Description

Return the list of correlated variables

Usage

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

Arguments

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

Value

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

See Also

cor

Examples

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

cost.curves *Plot Cost Curves*

Description

This function plots Cost Curves of several classification predictions.

Usage

```
cost.curves(
  predictions,
  gt,
  methods.names = NULL,
  positive = levels(factor(gt))[1],
  type = c("auto", "fuzzy", "hard"),
  ...
)
```

Arguments

predictions	The predictions of one or several classification models. Four shapes are accepted: a `factor` of hard labels (one model); a numeric `vector` of scores for the positive class (one model); a `matrix` of class probabilities, i.e. one column per class named after it, as returned by `predict(model, x, fuzzy = TRUE)` (one model); or any other `matrix`/`data.frame`, read as one column per model.
gt	Actual labels of the dataset (`factor` or `vector`), two classes only.
methods.names	The name of the compared methods (`vector`).
positive	The label of the positive class. Defaults to the first level of `gt`, as everywhere else in the package – note that for the usual alphabetical ordering this is often the *negative* class, so it is worth setting explicitly.
type	`"auto"` (default) reads `predictions` according to its shape, `"fuzzy"` requires scores and refuses hard labels, `"hard"` reduces everything to the predicted class first (this is the coarse three-point curve discussed above).
...	Other parameters, passed to the underlying plot.

Value

Nothing; the curves are drawn on the current graphics device.

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])
# From the estimated probabilities (the meaningful version)
cost.curves (predict (model.nb, d [, -5], fuzzy = TRUE), d [, 5])
# From hard labels, for comparison
cost.curves (cbind (predict (model.nb, d [, -5]), predict (model.lda, d [, -5])),
             d [, 5], c ("NB", "LDA"), type = "hard")
```

---

credit *Credit dataset*

Description

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

Usage

```
credit
```

Format

The dataset has 66 instances described by 11 qualitative variables.

`data.diag` *Square dataset*

Description

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

Usage

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

Arguments

`n`	Number of observations in the dataset.
`min`	Minimum value on each variables.
`max`	Maximum value on each variables.
`f`	The function that separates the classes.
`levels`	Name of each class.
`graph`	Whether the generated dataset is plotted. `FALSE` by default, as everywhere else in the package: a generator has to be callable in a loop or a report without piling up graphics devices.
`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 (graph = TRUE)
```

`data.gauss` *Gaussian mixture dataset*

Description

Generate a random multidimentional gaussian mixture.

Usage

```
data.gauss(
  n = 1000,
  k = 2,
  prob = rep(1/k, k),
  mu = cbind(rep(0, k), seq(from = 0, by = 3, length.out = k)),
  cov = rep(list(matrix(c(6, 0.9, 0.9, 0.3), ncol = 2, nrow = 2)), k),
  levels = NULL,
  graph = FALSE,
  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`	Whether the generated dataset is plotted. `FALSE` by default, as everywhere else in the package: a generator has to be callable in a loop or a report without piling up graphics devices.
`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 (graph = TRUE)
```

`data.parabol` *Parabol dataset*

Description

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

Usage

```
data.parabol(
  n = c(500, 100),
  xlim = c(-3, 3),
  center = c(0, 4),
  coeff = 0.5,
  sigma = c(0.5, 0.5),
  levels = NULL,
  graph = FALSE,
  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`	Whether the generated dataset is plotted. `FALSE` by default, as everywhere else in the package: a generator has to be callable in a loop or a report without piling up graphics devices.
`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 (graph = TRUE)
```

`data.target1` *Target1 dataset*

Description

Generate a random dataset shaped like a target.

Usage

```
data.target1(
  r = 1:3,
  n = 200,
  sigma = 0.1,
  levels = NULL,
  graph = FALSE,
  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`	Whether the generated dataset is plotted. `FALSE` by default, as everywhere else in the package: a generator has to be callable in a loop or a report without piling up graphics devices.
`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 (graph = TRUE)
```

`data.target2` *Target2 dataset*

Description

Generate a random dataset shaped like a target.

Usage

```
data.target2(
  minr = c(0, 2),
  maxr = minr + 1,
  initn = 1000,
  levels = NULL,
  graph = FALSE,
  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`	Whether the generated dataset is plotted. `FALSE` by default, as everywhere else in the package: a generator has to be callable in a loop or a report without piling up graphics devices.
`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 (graph = TRUE)
```

`data.twomoons` *Two moons dataset*

Description

Generate a random dataset shaped like two moons.

Usage

```
data.twomoons(
  r = 1,
  n = 200,
  sigma = 0.1,
  levels = NULL,
  graph = FALSE,
  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`	Whether the generated dataset is plotted. `FALSE` by default, as everywhere else in the package: a generator has to be callable in a loop or a report without piling up graphics devices.
`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 (graph = TRUE)
```

data.xor *XOR dataset*

Description

Generate "XOR" dataset.

Usage

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

Arguments

n	Number of observations in each cluster.
ndim	The number of dimensions (2^ndim clusters are formed, grouped into two classes).
sigma	The variance.
levels	Name of each class.
graph	Whether the generated dataset is plotted. FALSE by default, as everywhere else in the package: a generator has to be callable in a loop or a report without piling up graphics devices.
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 (graph = TRUE)
```

data1 *"data1" dataset*

Description

Synthetic dataset.

Usage

```
data1
```

Format

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

Author(s)

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

data2 *"data2" dataset*

Description

Synthetic dataset.

Usage

```
data2
```

Format

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

Author(s)

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

data3 *"data3" dataset*

Description

Synthetic dataset.

Usage

```
data3
```

Format

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

Author(s)

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

dataset-class *Training set and test set*

Description

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

Details

Objects of this class are plain lists with the following components:

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

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

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

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

See Also

splitdata

dbs-class *DBSCAN model*

Description

This class contains the model obtained by the DBSCAN method.

Details

Objects of this class are plain lists with the following components:

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

`eps` Reachability distance (parameter).

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

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

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

See Also

DBSCAN

DBSCAN *DBSCAN clustering method*

Description

Run the DBSCAN algorithm for clustering.

Usage

```
DBSCAN(d, minpts = 5, eps = NULL, graph = FALSE, ...)
```

Arguments

`d`	The dataset (`matrix` or `data.frame`).
`minpts`	Reachability minimum no. of points.
`eps`	Reachability distance. If `NULL` (the default), it is set to the knee of the sorted `minpts`-distance curve – the construction `distplot` invites you to do by eye – and the value used is reported in a message. This is only a starting point: `eps` is the parameter DBSCAN is most sensitive to, and it is worth looking at `distplot(minpts, d)` before settling on one.
`graph`	A logical indicating whether or not a graphic should be plotted (the `minpts`-distance curve used to choose `eps`, when `eps` is not given).
`...`	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 *Plot a k-distance graphic*

Description

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

Usage

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

Arguments

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

See Also

DBSCAN, dbscan

Examples

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

EM *Expectation-Maximization clustering method*

Description

Run the EM algorithm for clustering.

Usage

```
EM(d, k, model = "VVV", seed = NULL, ...)
```

Arguments

d	The dataset (matrix or data.frame).
k	Either an integer (the number of clusters) or a (vector) indicating the cluster to which each point is initially allocated.
model	A character string indicating the model. The help file for mclustModelNames describes the available models.
seed	A specified seed for random number generation (used only for the default k-means initialization).
...	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.

Details

Objects of this class are plain lists with the following components:

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

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

n The number of observations in the dataset.

d The number of variables in the dataset.

G The number of components of the mixture.

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

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

`control` A list of control parameters for EM.

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

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

See Also

`EM`, `mclustModelNames`

`eucalyptus` *Eucalyptus dataset*

Description

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

Usage

```
eucalyptus
```

Format

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

Source

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

`evaluation` *Evaluation of classification or regression predictions*

Description

Evaluation predictions of a classification or a regression model.

Usage

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

Arguments

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

Value

The evaluation of the predictions (numeric value).

See Also

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

Examples

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

---

evaluation.accuracy *Accuracy of classification predictions*

Description

Evaluation predictions of a classification model according to accuracy.

Usage

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

Arguments

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

Value

The evaluation of the predictions (numeric value).

See Also

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

Examples

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

---

evaluation.adjr2 *Adjusted R2 evaluation of regression predictions*

Description

Evaluation predictions of a regression model according to the adjusted R2, i.e. the R2 penalized by the number of variables used by the model.

Usage

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

Arguments

predictions	The predictions of a regression model (vector).
gt	The ground truth (vector).
nrow	Number of observations (defaults to the number of predictions).
ncol	Number of predictors used by the model. This one has no default: the adjustment cannot be computed without it. The residual degrees of freedom are nrow - ncol - 1, the intercept counting as one parameter; the value is NA when they run out.
...	Other parameters.

Value

The evaluation of the predictions (numeric value).

See Also

evaluation.r2, 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$test.x)
evaluation.adjr2 (pred.linreg, d$test.y, ncol = ncol (d$test.x))
```

---

evaluation.fmeasure *F-measure*

Description

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

Usage

```
evaluation.fmeasure(
  predictions,
  gt,
  beta = 1,
  average = NULL,
  positive = NULL,
  ...
)
```

Arguments

predictions	The predictions of a classification model (factor or vector).
gt	The ground truth (factor or vector).
beta	The weight given to precision.
average	How the per-class values are combined. These measures are defined for one class against all the others, so a single number requires either picking that class or averaging.
	"binary" the value for the class named by positive. Two-class problems only, and the default there.
	"macro" the plain mean of the per-class values. The default beyond two classes. Gives every class the same weight, whatever its size, so a rare class the model never gets right weighs as much as the majority one.

"weighted" the mean of the per-class values, weighted by the number of observations of each class.

"micro" pools the counts of every class before dividing. With single-label predictions each observation contributes one predicted and one actual label, so micro-averaged precision, recall and F-measure all equal the accuracy.

"none" the vector of per-class values, named after the classes. Cannot be used through evaluation or performance, which expect one number per criterion.

Averages run over the classes of the ground truth. A class that no observation is predicted to belong to has an undefined precision; it is read as 0, the usual convention.

positive — The label of the positive class, used by average = "binary" only. Defaults to levels (gt) [1], i.e. the *first* level of the ground truth factor – which, for the usual alphabetical level ordering, is often the *negative* class ("N" before "Y", "No" before "Yes", ...). Set this argument explicitly whenever the positive class is not the first level.

... — Other parameters.

Value

The evaluation of the predictions (numeric value).

See Also

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

Examples

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

---

evaluation.fowlkesmallows

Fowlkes–Mallows index

Description

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

Usage

```
evaluation.fowlkesmallows(
  predictions,
  gt,
  average = NULL,
  positive = NULL,
  ...
)
```

Arguments

predictions	The predictions of a classification model (factor or vector).
gt	The ground truth (factor or vector).
average	How the per-class values are combined. These measures are defined for one class against all the others, so a single number requires either picking that class or averaging. "binary" the value for the class named by positive. Two-class problems only, and the default there. "macro" the plain mean of the per-class values. The default beyond two classes. Gives every class the same weight, whatever its size, so a rare class the model never gets right weighs as much as the majority one. "weighted" the mean of the per-class values, weighted by the number of observations of each class. "micro" pools the counts of every class before dividing. With single-label predictions each observation contributes one predicted and one actual label, so micro-averaged precision, recall and F-measure all equal the accuracy. "none" the vector of per-class values, named after the classes. Cannot be used through evaluation or performance, which expect one number per criterion. Averages run over the classes of the ground truth. A class that no observation is predicted to belong to has an undefined precision; it is read as 0, the usual convention.
positive	The label of the positive class, used by average = "binary" only. Defaults to levels(gt)[1], i.e. the *first* level of the ground truth factor – which, for the usual alphabetical level ordering, is often the *negative* class ("N" before "Y", "No" before "Yes", ...). Set this argument explicitly whenever the positive class is not the first level.
...	Other parameters.

Value

The evaluation of the predictions (numeric value).

See Also

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

Examples

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

`evaluation.goodness` *Goodness*

Description

Evaluation predictions of a classification model according to Goodness index.

Usage

```
evaluation.goodness(
  predictions,
  gt,
  beta = 1,
  average = NULL,
  positive = NULL,
  ...
)
```

Arguments

`predictions`	The predictions of a classification model (`factor` or `vector`).
`gt`	The ground truth (`factor` or `vector`).
`beta`	The weight given to precision.
`average`	How the per-class values are combined. These measures are defined for one class against all the others, so a single number requires either picking that class or averaging. `"binary"` the value for the class named by `positive`. Two-class problems only, and the default there. `"macro"` the plain mean of the per-class values. The default beyond two classes. Gives every class the same weight, whatever its size, so a rare class the model never gets right weighs as much as the majority one. `"weighted"` the mean of the per-class values, weighted by the number of observations of each class. `"micro"` pools the counts of every class before dividing. With single-label predictions each observation contributes one predicted and one actual label, so micro-averaged precision, recall and F-measure all equal the accuracy.

"none" the vector of per-class values, named after the classes. Cannot be used through evaluation or performance, which expect one number per criterion.

Averages run over the classes of the ground truth. A class that no observation is predicted to belong to has an undefined precision; it is read as 0, the usual convention.

positive The label of the positive class, used by average = "binary" only. Defaults to levels(gt)[1], i.e. the *first* level of the ground truth factor – which, for the usual alphabetical level ordering, is often the *negative* class ("N" before "Y", "No" before "Yes", ...). Set this argument explicitly whenever the positive class is not the first level.

... Other parameters.

Value

The evaluation of the predictions (numeric value).

See Also

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

Examples

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

---

evaluation.jaccard *Jaccard index*

Description

Evaluation predictions of a classification model according to Jaccard index.

Usage

```
evaluation.jaccard(predictions, gt, average = NULL, positive = NULL, ...)
```

Arguments

predictions	The predictions of a classification model (factor or vector).
gt	The ground truth (factor or vector).
average	How the per-class values are combined. These measures are defined for one class against all the others, so a single number requires either picking that class or averaging. "binary" the value for the class named by positive. Two-class problems only, and the default there. "macro" the plain mean of the per-class values. The default beyond two classes. Gives every class the same weight, whatever its size, so a rare class the model never gets right weighs as much as the majority one. "weighted" the mean of the per-class values, weighted by the number of observations of each class. "micro" pools the counts of every class before dividing. With single-label predictions each observation contributes one predicted and one actual label, so micro-averaged precision, recall and F-measure all equal the accuracy. "none" the vector of per-class values, named after the classes. Cannot be used through evaluation or performance, which expect one number per criterion. Averages run over the classes of the ground truth. A class that no observation is predicted to belong to has an undefined precision; it is read as 0, the usual convention.
positive	The label of the positive class, used by average = "binary" only. Defaults to levels (gt) [1], i.e. the *first* level of the ground truth factor – which, for the usual alphabetical level ordering, is often the *negative* class ("N" before "Y", "No" before "Yes", ...). Set this argument explicitly whenever the positive class is not the first level.
...	Other parameters.

Value

The evaluation of the predictions (numeric value).

See Also

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

Examples

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

`evaluation.kappa` *Kappa evaluation of classification predictions*

Description

Evaluation predictions of a classification model according to Cohen's kappa: the proportion of correct predictions, corrected for the proportion two independent labellings would get right by chance. Class labels being nominal, the kappa is the unweighted one – every mistake counts the same.

Usage

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

Arguments

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

Value

The evaluation of the predictions (numeric value).

See Also

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

Examples

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

evaluation.msep *MSEP evaluation of regression predictions*

Description

Evaluation predictions of a regression model according to MSEP

Usage

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

Arguments

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

Value

The evaluation of the predictions (numeric value).

See Also

evaluation.r2, evaluation

Examples

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

evaluation.precision *Precision of classification predictions*

Description

Evaluation predictions of a classification model according to precision.

Usage

```
evaluation.precision(predictions, gt, average = NULL, positive = NULL, ...)
```

Arguments

predictions	The predictions of a classification model (factor or vector).
gt	The ground truth (factor or vector).
average	How the per-class values are combined. These measures are defined for one class against all the others, so a single number requires either picking that class or averaging. "binary" the value for the class named by positive. Two-class problems only, and the default there. "macro" the plain mean of the per-class values. The default beyond two classes. Gives every class the same weight, whatever its size, so a rare class the model never gets right weighs as much as the majority one. "weighted" the mean of the per-class values, weighted by the number of observations of each class. "micro" pools the counts of every class before dividing. With single-label predictions each observation contributes one predicted and one actual label, so micro-averaged precision, recall and F-measure all equal the accuracy. "none" the vector of per-class values, named after the classes. Cannot be used through evaluation or performance, which expect one number per criterion. Averages run over the classes of the ground truth. A class that no observation is predicted to belong to has an undefined precision; it is read as 0, the usual convention.
positive	The label of the positive class, used by average = "binary" only. Defaults to levels (gt) [1], i.e. the *first* level of the ground truth factor – which, for the usual alphabetical level ordering, is often the *negative* class ("N" before "Y", "No" before "Yes", ...). Set this argument explicitly whenever the positive class is not the first level.
...	Other parameters.

Value

The evaluation of the predictions (numeric value).

See Also

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

Examples

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

evaluation.r2 *R2 evaluation of regression predictions*

Description

Evaluation predictions of a regression model according to R2

Usage

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

Arguments

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

Value

The evaluation of the predictions (numeric value).

See Also

evaluation.msep, evaluation

Examples

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

evaluation.recall *Recall of classification predictions*

Description

Evaluation predictions of a classification model according to recall.

Usage

```
evaluation.recall(predictions, gt, average = NULL, positive = NULL, ...)
```

Arguments

predictions	The predictions of a classification model (factor or vector).
gt	The ground truth (factor or vector).
average	How the per-class values are combined. These measures are defined for one class against all the others, so a single number requires either picking that class or averaging. "binary" the value for the class named by positive. Two-class problems only, and the default there. "macro" the plain mean of the per-class values. The default beyond two classes. Gives every class the same weight, whatever its size, so a rare class the model never gets right weighs as much as the majority one. "weighted" the mean of the per-class values, weighted by the number of observations of each class. "micro" pools the counts of every class before dividing. With single-label predictions each observation contributes one predicted and one actual label, so micro-averaged precision, recall and F-measure all equal the accuracy. "none" the vector of per-class values, named after the classes. Cannot be used through evaluation or performance, which expect one number per criterion. Averages run over the classes of the ground truth. A class that no observation is predicted to belong to has an undefined precision; it is read as 0, the usual convention.
positive	The label of the positive class, used by average = "binary" only. Defaults to levels (gt) [1], i.e. the *first* level of the ground truth factor – which, for the usual alphabetical level ordering, is often the *negative* class ("N" before "Y", "No" before "Yes", ...). Set this argument explicitly whenever the positive class is not the first level.
...	Other parameters.

Value

The evaluation of the predictions (numeric value).

See Also

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

Examples

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

`exportgraphics` *Open a graphics device*

Description

Starts the graphics device driver

Usage

```
exportgraphics(
  file,
  type = tail(strsplit(file, split = "\\.")[[1]], 1),
  export = fdm2id.globals$export,
  ...
)
```

Arguments

`file`	A character string giving the name of the file.
`type`	The type of graphics device. Deduced from the file extension by default: `"eps"` and `"ps"` go to `postscript`, `"jpg"` to `jpeg`, and any other extension is taken as the name of an R function (`"pdf"`, `"png"`, ...).
`export`	If given, explicitly overrides the global export toggle set by `toggleexport` for this call only. By default, the global toggle is used, so existing code is unaffected.
`...`	Other parameters.

See Also

`closegraphics`, `toggleexport`, `Devices`

Examples

```
## Not run:
data (iris)
exportgraphics ("export.pdf")
plotdata (iris [, -5], iris [, 5])
closegraphics()
# Extensions that don't match an R function name directly are now handled:
exportgraphics ("export.eps")
plotdata (iris [, -5], iris [, 5])
closegraphics()

## End(Not run)
```

`exportgraphics.off` *Toggle graphic exports*

Description

Toggle graphic exports on and off

Usage

```
exportgraphics.off()

exportgraphics.on()

toggleexport(export = NULL)

toggleexport.off()

toggleexport.on()
```

Arguments

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

See Also

`closegraphics`, `exportgraphics`

Examples

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

## End(Not run)
```

factorial-class *Factorial analysis results*

Description

This class contains the result of a factorial analysis, as obtained by CA, MCA or PCA.

Details

Objects of this class are the objects returned by the corresponding **FactoMineR** functions (CA, MCA and PCA), with the extra class factorial prepended so that plot.factorial can provide a uniform plotting interface. The second class ("ca", "mca" or "pca") records which analysis was performed.

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") fseval.univariate() else NULL,
  uninb = NULL,
  unithreshold = NULL,
  multieval = fseval.multivariate(),
  wrapmethod = NULL,
  mainmethod = wrapmethod,
  tune = FALSE,
  methodparameters = NULL,
  graph = FALSE,
  seed = NULL,
  ...
)
```

Arguments

train	The training set (description), as a data.frame.
labels	Class labels of the training set (vector or factor).
algorithm	The feature selection algorithm.
unieval	The (univariate) evaluation criterion. uninb, unithreshold or multieval must be specified.
uninb	The number of selected feature (univariate evaluation).
unithreshold	The threshold for selecting feature (univariate evaluation).
multieval	The (multivariate) evaluation criterion.
wrapmethod	The classification method used for the wrapper evaluation.
mainmethod	The final method used for data classification (required: either mainmethod or wrapmethod must be a valid classification/regression function, e.g. LDA, NB, ...). If a wrapper evaluation is used, the same classification method should be used.
tune	If true, the function returns parameters instead of a classification model.
methodparameters	Pre-tuned parameters, as returned by the same method called with tune = TRUE. performance obtains them once and passes them back when fitting, so that the tuning is not redone on every split. A method with nothing to tune returns an empty object, which leaves its defaults untouched.
graph	Whether the method draws the graphic that goes with its tuning (the cross-validation curve, typically). Methods that have no such graphic accept the argument and ignore it.
seed	A specified seed for random number generation, so that two runs on the same data give the same model. Every learning method accepts it, so that it can be set the same way whatever the method; the deterministic ones simply have nothing to draw and give the same model with or without it.
...	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, parameter = list (supp = .4, conf = .8))
inspect (filter.rules (r, right = "marital-status="))
# The equivalent call in arules itself
subset (r, subset = rhs %pin% "marital-status=")
```

frequentwords *Frequent words*

Description

Most frequent words of the corpus.

Usage

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

Arguments

corpus	The corpus of documents (a vector of characters) or the vocabulary of the documents (result of function getvocab).
nb	The number of words to be returned.
mincount	Minimum word count to be considered as frequent.
minphrasecount	Minimum collocation of words count to be considered as frequent.
ngram	maximum size of n-grams.
lang	The language of the documents (NULL if no stemming).
stopwords	The language whose stop words are removed ("en", ...), or NULL to keep them. A list of words of your own goes to excludewords.
excludewords	An optional custom vector of additional words to exclude from the vocabulary (e.g. corpus-specific stop words), on top of (or instead of) the language stopwords given through stopwords.
removesinglechars	Whether single-character tokens are removed during cleanup.

Value

The most frequent words of the corpus.

See Also

getvocab

Examples

```
data (capitals)
frequentwords (capitals, 10, mincount = 2)
vocab = getvocab (capitals, mincount = 2)
frequentwords (vocab, 10)
```

GBREG *Regression using Gradient Boosting*

Description

This function builds a regression model using Gradient Boosting. It is the regression counterpart of GRADIENTBOOSTING, which classifies.

Usage

```
GBREG(
  x,
  y,
  ntree = 500,
  learningrate = 0.3,
  tune = FALSE,
  methodparameters = NULL,
  graph = FALSE,
  seed = NULL,
  ...
)
```

Arguments

x	Predictor values of the training set, as a matrix or data.frame.
y	Target values of the training set (a numeric vector).
ntree	The number of trees in the ensemble.
learningrate	The learning rate (between 0 and 1).
tune	If true, the function returns parameters instead of a classification model.
methodparameters	Present for interface consistency with performance (which always passes it when fitting a model). Currently unused: GBREG does not yet implement hyperparameter tuning.
graph	Present for interface consistency with performance (which always passes it when fitting a model). Currently unused: GBREG does not produce a plot.
seed	A specified seed for random number generation (row/column subsampling, if used via ...).
...	Other parameters, passed to xgboost.

Value

The regression model.

See Also

GRADIENTBOOSTING, LINREG, SVR, xgboost

Examples

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

---

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

Description

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

Usage

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

Arguments

r	A set of rules.

Value

A set of rules, without redundancy.

See Also

apriori

Examples

```
require ("arules")
data ("Adult")
# The default support (0.1) yields ~6000 rules on Adult, and general.rules() compares every
# pair of them twice: that single call took more than 8 seconds. A higher support keeps the
# example instructive (169 rules, of which 8 are general) and instantaneous.
r = apriori (Adult, parameter = list (supp = .4, conf = .8))
inspect (general.rules (r))
```

getvocab *Extract words and phrases from a corpus*

Description

Extract words and phrases from a corpus of documents.

Usage

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

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	The language whose stop words are removed ("en", ...), or NULL to keep them. A list of words of your own goes to excludewords.
excludewords	An optional custom vector of additional words to exclude from the vocabulary (e.g. corpus-specific stop words), on top of (or instead of) the language stopwords given through stopwords.
removesinglechars	Whether single-character tokens are removed during cleanup.
...	Other parameters.

Value

The vocabulary used in the corpus of documents.

See Also

plotzipf, stopwords, create_vocabulary

Examples

```
data (capitals)
vocab1 = getvocab (capitals, mincount = 2) # With stemming
nrow (vocab1)
vocab2 = getvocab (capitals, mincount = 2, lang = NULL) # Without stemming
nrow (vocab2)
# Excluding additional, corpus-specific words
vocab3 = getvocab (capitals, mincount = 2, excludewords = c ("capital", "europe"))
```

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,
  methodparameters = NULL,
  graph = FALSE,
  seed = NULL,
  ...
)
```

Arguments

train	The training set (description), as a data.frame.
labels	Class labels of the training set (vector or factor).
ntree	The number of trees in the forest.
learningrate	The learning rate (between 0 and 1).
tune	If true, the function returns parameters instead of a classification model.
methodparameters	Present for interface consistency with performance (which always passes it when fitting a model). Currently unused: GRADIENTBOOSTING does not yet implement hyperparameter tuning.
graph	Present for interface consistency with performance (which always passes it when fitting a model). Currently unused: GRADIENTBOOSTING does not produce a plot.

seed	A specified seed for random number generation (row/column subsampling, if used via ...; xgboost's default parameters are otherwise deterministic, but the seed is provided for consistency with the rest of the package's API and to cover subsampling parameters passed through ...).
...	Other parameters.

Value

The classification model.

See Also

xgboost

Examples

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

HCA *Hierarchical Cluster Analysis method*

Description

Run the HCA method for clustering.

Usage

```
HCA(
  d,
  k = NULL,
  method = c("ward", "single"),
  engine = c("hclust", "agnes"),
  graph = FALSE,
  ...
)
```

Arguments

d	The dataset (matrix or data.frame).
k	The number of cluster. If NULL (the default), it is set to the largest drop in aggregation height.
method	Character string defining the clustering method.

engine	Which implementation builds the hierarchy: hclust (the default) or agnes. They give the same hierarchy – same heights, to machine precision, and the same cut – but hclust is far faster on a large dataset (0.3 s against 30 s on 3000 observations). Use "agnes" for the linkages it alone provides, or to compare the two.
graph	A logical indicating whether or not a graphic should be plotted (the aggregation heights used to choose k, when k is not given).
...	Other parameters.

Value

The cluster hierarchy (hca object).

See Also

hclust, agnes, treeplot, predict.hca

Examples

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

---

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$cluster, iris [, -5])
intern (km$cluster, iris [, -5], type = "cluster")
intern (km$cluster, iris [, -5], eval = c ("intraclass", "interclass"))
intern (km$cluster, 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", "cluster"))
```

Arguments

clus	The extracted clusters.
d	The dataset.
type	Indicates whether a "global" or a "cluster"-wise evaluation should be used. The per-cluster values are the terms the global index is the minimum of: each cluster's distance to the nearest other one, over the largest diameter of the partition.

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$cluster, iris [, -5])
intern.dunn (km$cluster, iris [, -5], type = "cluster")
```

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$cluster, iris [, -5])
```

intern.intraclass *Clustering evaluation through intraclass inertia*

Description

Evaluation a clustering algorithm according to intraclass inertia.

Usage

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

Arguments

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

Value

The evaluation of the clustering.

See Also

intern, intern.dunn, intern.interclass

Examples

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

---

ionosphere *Ionosphere dataset*

Description

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

Usage

```
ionosphere
```

Format

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

Source

https://archive.ics.uci.edu/dataset/52/ionosphere

kaiser *Kaiser rule*

Description

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

Usage

```
kaiser(pca)
```

Arguments

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

See Also

PCA, factorial-class

Examples

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

KERREG *Kernel Regression*

Description

This function builds a kernel regression model.

Usage

```
KERREG(
  x,
  y,
  bandwidth = 1,
  tune = FALSE,
  methodparameters = NULL,
  graph = FALSE,
  seed = NULL,
  ...
)
```

Arguments

x	Predictor matrix.
y	Response vector.
bandwidth	The bandwidth parameter.
tune	If true, the function returns parameters instead of a classification model.
methodparameters	Present for interface consistency with performance (which always passes it when fitting a model). Currently unused: KERREG does not support reusing pre-tuned parameters.
graph	Present for interface consistency with performance (which always passes it when fitting a model). Currently unused: KERREG does not produce a plot.
seed	A specified seed for random number generation, so that two runs on the same data give the same model. Every learning method accepts it, so that it can be set the same way whatever the method; the deterministic ones simply have nothing to draw and give the same model with or without it.
...	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", "silhouette", "gap", "elbow"),
  nstart = 10,
```

```
  B = 100,
  graph = FALSE,
  seed = NULL,
  ...
)
```

Arguments

d	The dataset (matrix or data.frame).
k	The number of cluster.
criterion	How the number of clusters is chosen: "none" (the default, use k as it is), "pseudo-F", "silhouette", "gap" or "elbow". With any but the first, k is read as the *largest* number of clusters to consider. See the Details section.
nstart	Define how many random sets should be chosen.
B	The number of bootstrap samples used by criterion = "gap".
graph	A logical indicating whether or not a graphic should be plotted (cluster number selection).
seed	A specified seed for random number generation. *K*-means starts from a random initialisation, so without a seed two calls on the same data give different clusterings; every other clustering function of the package already had this parameter.
...	Other parameters.

Details

The four criteria criterion offers, all computed between 2 clusters and k:

"pseudo-F" the Calinski-Harabasz index, between-cluster over within-cluster variance corrected for the number of clusters. Maximised.

"silhouette" the mean silhouette width – how much closer each observation is to its own cluster than to the nearest other one. Maximised.

"gap" the gap statistic: the distance between the observed within-cluster dispersion and the one expected with no cluster structure at all. The retained k is the smallest whose gap is within one standard error of the next. It is the only criterion that can answer k = 1, and much the slowest, needing B bootstrap samples.

"elbow" the bend of the total within-cluster sum of squares. That quantity decreases with k whatever the data, so there is no optimum to take: the retained k is the point furthest from the chord joining the two ends of the curve, drawn on the graphic.

The last three need the **cluster** package.

Value

The clustering (kmeans object).

See Also

kmeans, predict.kmeans

Examples

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

---

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

Description

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

Usage

```
kmeans.getk(
  d,
  max = 9,
  criterion = c("pseudo-F", "silhouette", "gap", "elbow"),
  nstart = 10,
  B = 100,
  graph = FALSE,
  seed = NULL
)
```

Arguments

d	The dataset (matrix or data.frame).
max	The largest number of clusters considered. Values from 2 to max are evaluated (from 1, for "gap" and "elbow", which are defined there).
criterion	How the number of clusters is chosen: "none" (the default, use k as it is), "pseudo-F", "silhouette", "gap" or "elbow". With any but the first, k is read as the *largest* number of clusters to consider. See the Details section.
nstart	The number of random sets chosen for kmeans initialization.
B	The number of bootstrap samples used by criterion = "gap".
graph	A logical indicating whether or not a graphic should be plotted.
seed	A specified seed for random number generation.

Value

The number of clusters retained by the chosen criterion.

References

Tibshirani, R., Walther, G. and Hastie, T. (2001). Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society: Series B*, 63(2), 411-423.

See Also

pseudoF, KMEANS, kmeans, silhouette, clusGap

Examples

```
require (datasets)
data (iris)
kmeans.getk (iris [, -5])
kmeans.getk (iris [, -5], criterion = "silhouette")
kmeans.getk (iris [, -5], criterion = "elbow")

# The gap statistic resamples, so it is much slower than the other three.
kmeans.getk (iris [, -5], criterion = "gap", B = 20, seed = 0)
```

KNN *Classification using k-NN*

Description

This function builds a classification model using Logistic Regression.

Usage

```
KNN(
  train,
  labels,
  k = 1:10,
  nfolds = 10,
  tune = FALSE,
  methodparameters = NULL,
  graph = FALSE,
  seed = NULL,
  ...
)
```

Arguments

train	The training set (description), as a data.frame.
labels	Class labels of the training set (vector or factor).
k	The k parameter.
nfolds	The number of folds of the cross-validation a method runs to choose its hyperparameters. Only used when there is something to choose, i.e. when one of them is given as a vector. Lower it to fit faster, at the cost of a noisier choice.
tune	If true, the function returns parameters instead of a classification model.

`methodparameters`
: Present for interface consistency with `performance` (which always passes it when fitting a model). Currently unused: `KNN` does not support reusing pre-tuned parameters (it stores the training set and re-tunes `k` on every call when `k` is a vector).

`graph`
: Present for interface consistency with `performance` (which always passes it when fitting a model). Currently unused: `KNN` does not produce a plot.

`seed`
: A specified seed for random number generation, so that two runs on the same data give the same model. Every learning method accepts it, so that it can be set the same way whatever the method; the deterministic ones simply have nothing to draw and give the same model with or without it.

`...`
: 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.

Details

Objects of this class are plain lists with the following components:

`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,
  methodparameters = NULL,
  graph = FALSE,
  seed = NULL,
  ...
)
```

Arguments

train	The training set (description), as a data.frame.
labels	Class labels of the training set (vector or factor).
tune	If true, the function returns parameters instead of a classification model.
methodparameters	Present for interface consistency with performance (which always passes it when fitting a model). Currently unused: LDA does not support reusing pre-tuned parameters.
graph	Present for interface consistency with performance (which always passes it when fitting a model). Currently unused: LDA does not produce a plot.
seed	A specified seed for random number generation, so that two runs on the same data give the same model. Every learning method accepts it, so that it can be set the same way whatever the method; the deterministic ones simply have nothing to draw and give the same model with or without it.
...	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 the x-axis.
labels	The labels of the instances.

Examples

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

LINREG *Linear Regression*

Description

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

Usage

```
LINREG(
  x,
  y,
  quali = c("none", "intercept", "slope", "both"),
  reg = c("linear", "subset", "ridge", "lasso", "elastic", "pcr", "plsr"),
 regeval = if (reg[1] == "subset") c("bic", "adjr2", "cp", "r2") else c("r2", "msep"),
  scale = TRUE,
  validation = c("CV", "LOO"),
  lambda = 10^seq(-5, 5, length.out = 101),
  alpha = 0.5,
  nrep = 1,
  tune = FALSE,
  methodparameters = NULL,
  graph = FALSE,
  seed = NULL,
  ...
)
```

Arguments

x	Predictor matrix.
y	Response vector.
quali	Indicates how to use the qualitative variables.
reg	The algorithm.
regeval	The criterion used to choose between models. For reg = "subset": "bic" (the default), "adjr2" or "cp", which all penalize the number of variables, and "r2", which does not – the R2 can only grow when a variable is added, so it always retains every variable. For reg = "pcr" and reg = "plsr", where the choice is a number of components: "r2" (the default) or "msep". Ignored by the other algorithms.
scale	If true, PCR and PLS use scaled dataset.
validation	How the number of components of a PCR or PLS regression is chosen: "CV" (10 random segments, the default) or "LOO" (leave-one-out, one regression per observation – only practical on a small dataset). Ignored by the other algorithms.
lambda	The lambda parameter of Ridge, Lasso and Elastic net regression.
alpha	The elasticnet mixing parameter.
nrep	How many times the cross-validation choosing lambda is repeated, its errors being averaged. One is enough in practice – cv.glmnet already averages over its own folds – and each extra repetition costs another ten fits. Raise it to steady the choice of lambda on a small or noisy dataset.
tune	If true, the function returns parameters instead of a classification model.
methodparameters	Present for interface consistency with performance (which always passes it when fitting a model). Currently unused: LINREG does not support reusing pre-tuned parameters.

graph	A logical indicating whether or not graphics should be plotted (ridge, LASSO and elastic net).
seed	A specified seed for random number generation, so that two runs on the same data give the same model. Every learning method accepts it, so that it can be set the same way whatever the method; the deterministic ones simply have nothing to draw and give the same model with or without it.
...	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 independent variable
data (cars)
LINREG (cars [, -2], cars [, 2])
# With two independent variables
data (trees)
LINREG (trees [, -3], trees [, 3])
# With non numeric variables
data (ToothGrowth)
LINREG (ToothGrowth [, -1], ToothGrowth [, 1], quali = "intercept") # Different intercept
LINREG (ToothGrowth [, -1], ToothGrowth [, 1], quali = "slope") # Different slope
LINREG (ToothGrowth [, -1], ToothGrowth [, 1], quali = "both") # Complete model
# With multiple numeric variables
data (mtcars)
LINREG (mtcars [, -1], mtcars [, 1])
LINREG (mtcars [, -1], mtcars [, 1], reg = "subset", regeval = "adjr2")
LINREG (mtcars [, -1], mtcars [, 1], reg = "ridge")
LINREG (mtcars [, -1], mtcars [, 1], reg = "lasso")
LINREG (mtcars [, -1], mtcars [, 1], reg = "elastic")
LINREG (mtcars [, -1], mtcars [, 1], reg = "pcr")
LINREG (mtcars [, -1], mtcars [, 1], reg = "plsr")

## End(Not run)
```

---

linsep *Linsep dataset*

Description

Synthetic dataset.

Usage

```
linsep
```

Format

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

Author(s)

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

`loadtext` *load a text file*

Description

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

Usage

```
loadtext(
  file = NULL,
  dir = tempdir(),
  collapse = TRUE,
  sep = NULL,
  categories = NULL,
  cache = FALSE
)
```

Arguments

`file`	The path or URL of the text file. If not specified, defaults to an interactive file chooser (`file.choose`) when running interactively; in a non-interactive session (script, CI, `R CMD check`), `file` must be given explicitly.
`dir`	The directory the file is downloaded (and, for a zip archive, extracted) into. Defaults to the session's temporary directory, which is emptied when R exits. Pass an explicit path (together with `cache = TRUE`) to keep the downloaded corpus between sessions.
`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 categorical data.
`cache`	Whether the downloaded (and, for a zip archive, extracted) files are kept in `dir` and reused on the next call. They are deleted, and downloaded again every time, by default.

Value

The text contained in the dowloaded file.

See Also

download.file, unzip

Examples

```
# Not run automatically: this downloads a 31 MB archive from a third-party server, so it
# depends on both the network and that server staying up.
## Not run:
text = loadtext ("http://mattmahoney.net/dc/text8.zip")
# Keep the archive between calls, in a directory of your choosing
text = loadtext ("http://mattmahoney.net/dc/text8.zip", dir = "~/corpora", cache = TRUE)

## End(Not run)
```

LR *Classification using Logistic Regression*

Description

This function builds a classification model using Logistic Regression.

Usage

```
LR(
  train,
  labels,
  reg = c("none", "ridge", "lasso", "elastic"),
  lambda = NULL,
  alpha = 0.5,
  nfolds = 10,
  tune = FALSE,
  methodparameters = NULL,
  graph = FALSE,
  seed = NULL,
  ...
)
```

Arguments

train	The training set (description), as a data.frame.
labels	Class labels of the training set (vector or factor).

reg — The penalty applied to the coefficients, as in LINREG: "none" (the default) fits the plain multinomial logistic regression of multinom, while "ridge" (L2), "lasso" (L1) and "elastic" (a mix of the two, weighted by alpha) fit a penalized one with glmnet. Penalizing is what makes logistic regression usable when the predictors are numerous or strongly correlated, where the unpenalized fit either fails to converge or separates the classes perfectly with unbounded coefficients.

lambda — The grid of penalty strengths searched by cross-validation; the retained value is the one minimising the cross-validated deviance. NULL (the default) lets glmnet derive the grid from the data, which is the recommended choice: a fixed grid reaching very small penalties makes the fit fail to converge on separable data.

alpha — The elastic net mixing parameter, between 0 (ridge) and 1 (lasso). Used by reg = "elastic" only.

nfolds — The number of folds of the cross-validation used to choose lambda.

tune — If true, the function returns parameters instead of a classification model.

methodparameters — Present for interface consistency with performance (which always passes it when fitting a model). Currently unused: LR does not support reusing pre-tuned parameters.

graph — Whether the cross-validation curve used to choose lambda is plotted. Ignored by reg = "none", which has nothing to choose.

seed — A specified seed for random number generation, so that two runs on the same data give the same model. Every learning method accepts it, so that it can be set the same way whatever the method; the deterministic ones simply have nothing to draw and give the same model with or without it.

... — Other parameters.

Value

The classification model.

See Also

multinom, glmnet, LINREG

Examples

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

# Penalized variants: same three penalties as LINREG()
d = splitdata (iris, 5, seed = 0)
model = LR (d$train.x, d$train.y, reg = "lasso")
evaluation (predict (model, d$test.x), d$test.y)
```

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

Arguments

d	A data frame or a table with n rows and p columns, i.e. a contingency table.
ncp	The number of dimensions kept in the results. All of them, by default: one per active variable, and never more than $n - 1$. A variable with J_q modalities contributes $J_q - 1$ of them – its indicator columns sum to one, so one of them is redundant – which makes $J - Q$ for Q active variables holding J modalities between them, and never more than $n - 1$. The modalities are counted as *observed* among the active individuals: a level nobody takes carries nothing.
ind.sup	A vector indicating the indexes of the supplementary individuals.
quanti.sup	A vector indicating the indexes of the quantitative supplementary variables.
quali.sup	A vector indicating the indexes of the categorical supplementary variables.
row.w	An optional row weights (by default, a vector of 1 for uniform row weights); the weights are given only for the active individuals.

Value

The MCA on the dataset.

See Also

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

Examples

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

MEANSHIFT *MeanShift method*

Description

Run MeanShift for clustering.

Usage

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

Arguments

d	The dataset (matrix or data.frame).
mskernel	A string indicating the kernel associated with the kernel density estimate that the mean shift is optimizing over.
bandwidth	Used in the kernel density estimate for steepest ascent classification.
alpha	A scalar tuning parameter for normal kernels.
iterations	The number of iterations to perform mean shift.
epsilon	A scalar used to determine when to terminate the iteration of an individual query point.
epsilonCluster	A scalar used to determine the minimum distance between distinct clusters.
seed	A specified seed for random number generation. The MeanShift algorithm itself is deterministic given its parameters, but the seed is provided for consistency with the rest of the package's API.
...	Other parameters.

Value

The clustering (meanshift object).

See Also

meanShift, predict.meanshift

Examples

```
require (datasets)
data (iris)
MEANSHIFT (iris [, -5], bandwidth = .75)
```

---

meanshift-class *MeanShift model*

Description

This class contains the model obtained by the MEANSHIFT method.

Details

Objects of this class are plain lists with the following components:

`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 an individual query point.

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

See Also

MEANSHIFT

MLP *Classification using Multilayer Perceptron*

Description

This function builds a classification model using Multilayer Perceptron.

Usage

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

Arguments

train	The training set (description), as a data.frame.
labels	Class labels of the training set (vector or factor).
hidden	The size of the hidden layer (if a vector, cross-over validation is used to chose the best size).
decay	The decay (between 0 and 1) of the backpropagation algorithm (if a vector, cross-over validation is used to chose the best size).
nfolds	The number of folds of the cross-validation a method runs to choose its hyperparameters. Only used when there is something to choose, i.e. when one of them is given as a vector. Lower it to fit faster, at the cost of a noisier choice.
tune	If true, the function returns parameters instead of a classification model.
methodparameters	Object containing the parameters. If given, it replaces size and decay.
graph	Present for interface consistency with performance (which always passes it when fitting a model). Currently unused: MLP does not produce a plot.
seed	A specified seed for random number generation, so that two runs on the same data give the same model. Every learning method accepts it, so that it can be set the same way whatever the method; the deterministic ones simply have nothing to draw and give the same model with or without it.
...	Other parameters.

Value

The classification model.

See Also

nnet

Examples

```
require (datasets)
data (iris)
MLP (iris [, -5], iris [, 5], hidden = 4, decay = .1)
```

MLPREG *Multi-Layer Perceptron Regression*

Description

This function builds a regression model using MLP.

Usage

```
MLPREG(
  x,
  y,
  size = if (is.vector(x)) 2 else 2:ncol(x),
  decay = 10^(-3:-1),
  nfolds = 10,
  tune = FALSE,
  methodparameters = NULL,
  graph = FALSE,
  seed = NULL,
  ...
)
```

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).
nfolds	The number of folds of the cross-validation a method runs to choose its hyperparameters. Only used when there is something to choose, i.e. when one of them is given as a vector. Lower it to fit faster, at the cost of a noisier choice.

tune	If true, the function returns parameters instead of a classification model.
methodparameters	Object containing the parameters. If given, it replaces size and decay. Named (and behaves identically to) methodparameters rather than params, for consistency with MLP and with the calling convention used by performance/the internal protocol.* functions, which always pass a methodparameters argument.
graph	Present for interface consistency with performance (which always passes it when fitting a model). Currently unused: MLPREG does not produce a plot.
seed	A specified seed for random number generation, so that two runs on the same data give the same model. Every learning method accepts it, so that it can be set the same way whatever the method; the deterministic ones simply have nothing to draw and give the same model with or without it.
...	Other parameters.

Value

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

See Also

nnet

Examples

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

## End(Not run)
```

---

model-class *Generic classification or regression model*

Description

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

Details

Objects of this class are plain lists with the following components:

model The wrapped model.

method The name of the method.

See Also

predict.model, predict

movies *Movies dataset*

Description

Simulated ratings of 49 real films by 55 imaginary viewers.

Usage

```
movies
```

Format

A `matrix` of 49 films by 55 viewers. Ratings are whole numbers from 1 to 10.

Source

The structure was estimated on the MovieLens 100K dataset (https://grouplens.org/datasets/movielens/100k/), whose terms do not permit redistributing the ratings themselves.

NB *Classification using Naive Bayes*

Description

This function builds a classification model using Naive Bayes.

Usage

```
NB(
  train,
  labels,
  tune = FALSE,
  methodparameters = NULL,
  graph = FALSE,
  seed = NULL,
  ...
)
```

Arguments

train	The training set (description), as a data.frame.
labels	Class labels of the training set (vector or factor).
tune	If true, the function returns parameters instead of a classification model.
methodparameters	Present for interface consistency with performance (which always passes it when fitting a model). Currently unused: NB does not support reusing pre-tuned parameters.
graph	Present for interface consistency with performance (which always passes it when fitting a model). Currently unused: NB does not produce a plot.
seed	A specified seed for random number generation, so that two runs on the same data give the same model. Every learning method accepts it, so that it can be set the same way whatever the method; the deterministic ones simply have nothing to draw and give the same model with or without it.
...	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, seed = NULL, ...)
```

Arguments

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

See Also

nmf

Examples

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

## End(Not run)
```

---

ozone *Ozone dataset*

Description

This dataset constains measurements on ozone level.

Usage

```
ozone
```

Format

112 days described by 13 variables. maxO3, the maximum level of ozone measured during the day (the target variable); T9, T12, T15, temperatures; C9, C12, C15, cloud cover; W9, W12, W15, the projection of the wind on the North-South axis; maxO3v, the maximum level of ozone of the previous day; vent, the wind direction (a factor: "Est", "Nord", "Ouest", "Sud"); pluie, whether it rained (a factor: "Pluie", "Sec").

Source

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

PAM *Clustering using* K*-medoids (PAM)*

Description

Partitions the data into k clusters around *medoids* – actual observations of the dataset – rather than around means. Being an observation, a medoid can be shown to students as a representative example of its cluster, and the method tolerates outliers much better than *K*-means, which drags a mean towards them.

Usage

```
PAM(
  d,
  k = 9,
  criterion = c("none", "silhouette"),
  graph = FALSE,
  seed = NULL,
  ...
)
```

Arguments

d	The dataset (matrix or data.frame).
k	The number of clusters.
criterion	How k is chosen, as in KMEANS. With "none" (the default) k is used as it is; with "silhouette" it is chosen between 2 and k by the mean silhouette width, the criterion PAM itself optimises the closest.
graph	A logical indicating whether the criterion curve is plotted.
seed	A specified seed for random number generation. PAM's initialisation is deterministic, so this only matters for the criterion search.
...	Other parameters, passed to pam.

Value

The clustering, as an object of class pam (see pam), with a cluster component holding the assignments and a medoids one holding the representative observations.

See Also

KMEANS, pam, kmeans.getk

Examples

```
require (datasets)
data (iris)
model = PAM (iris [, -5], 3)
model$medoids
table (model$cluster, iris [, 5])
```

`params-class` *Learning Parameters*

Description

This class contains main parameters for various learning methods.

Details

Objects of this class are plain lists with the following components:

`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 = pca.ncp(d, ind.sup, quanti.sup, quali.sup),
  ind.sup = NULL,
  quanti.sup = NULL,
  quali.sup = NULL,
  row.w = NULL,
  col.w = NULL
)
```

Arguments

d	A data frame with n rows (individuals) and p columns (numeric variables).
scale.unit	A boolean, if TRUE (value set by default) then data are scaled to unit variance.
ncp	The number of dimensions kept in the results. All of them, by default: one per active variable, and never more than $n - 1$.
ind.sup	A vector indicating the indexes of the supplementary individuals.
quanti.sup	A vector indicating the indexes of the quantitative supplementary variables.
quali.sup	A vector indicating the indexes of the categorical supplementary variables.
row.w	An optional row weights (by default, a vector of 1 for uniform row weights); the weights are given only for the active individuals.
col.w	An optional column weights (by default, uniform column weights); the weights are given only for the active variables.

Value

The PCA on the dataset.

See Also

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

Examples

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

`performance` *Performance estimation*

Description

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

Usage

```
performance(
  methods,
  train.x,
  train.y,
  test.x = NULL,
  test.y = NULL,
  train.size = round(0.7 * nrow(train.x)),
  type = c("evaluation", "confusion", "roc", "cost", "scatter", "avsp"),
  protocol = c("bootstrap", "crossvalidation", "loocv", "holdout", "train"),
  eval = ifelse(is.factor(train.y), "accuracy", "r2"),
  nruns = 10,
  nfolds = 10,
  new = TRUE,
  lty = 1,
  seed = NULL,
  methodparameters = NULL,
  names = NULL,
  fuzzy = FALSE,
  positive = NULL,
  stratify = TRUE,
  ...
)
```

Arguments

`methods`	The classification or regression methods to be evaluated.
`train.x`	The dataset (description/predictors), a `matrix` or `data.frame`.
`train.y`	The target (class labels or numeric values), a `factor` or `vector`.
`test.x`	The test dataset (description/predictors), a `matrix` or `data.frame`. Giving `test.x` and `test.y` is the simplest use of this function: each method is fitted on `(train.x, train.y)`, used to predict `test.x`, and its predictions are compared with `test.y` – no resampling at all. `protocol` then defaults to `"holdout"`, and any other protocol raises an error, since it would ignore the test set.
`test.y`	The (test) target (class labels or numeric values), a `factor` or `vector`.
`train.size`	The size of the training set, for `protocol = "holdout"` without an explicit test set: either a number of observations, or a proportion between 0 and 1.

`type` The type of evaluation (confusion matrix, ROC curve, ...)

`protocol` How the performance is estimated.

`"bootstrap"` (default) `nruns` bootstrap samples of the training set, each model evaluated on the observations left out of its sample.

`"crossvalidation"` `nruns` repetitions of a `nfolds`-fold cross-validation of the training set.

`"loocv"` leave-one-out cross-validation.

`"holdout"` a single train/test split. Uses `test.x`/`test.y` when they are given – see `test.x` – and otherwise draws a training set of `train.size` observations and evaluates on the rest.

`"train"` evaluates each model on the very data it was fitted on. Optimistic by construction; useful to show students exactly that.

`eval` The evaluation functions.

`nruns` The number of bootstrap runs.

`nfolds` The number of folds (crossvalidation estimation).

`new` A logical value indicating whether a new plot should be created or not (cost curves or ROC curves).

`lty` The line type (and color) specified as an integer (cost curves or ROC curves).

`seed` A specified seed for random number generation (useful for testing different method with the same bootstap samplings).

`methodparameters`
Method parameters (if null tuning is done by cross-validation).

`names` Method names.

`fuzzy` Used by `type = "roc"` and `type = "cost"` only. `FALSE` by default: the curves are built from the hard class labels, which reduces them to three points. Pass `fuzzy = TRUE` to build them from the estimated probabilities of the positive class, which is what a ROC curve is meant to show. See `roc.curves`.

`positive` The label of the positive class. Used by `type = "roc"` and `type = "cost"` to orient the curves, and passed on to `evaluation` for the criteria that are defined on one class – precision, recall, the F-measure and the other measures taking an `average` argument – so that a two-class problem can be scored on either of its classes. Defaults to the first level of the target, which is worth setting explicitly whenever the class of interest is not the first one.

`stratify` Whether the splits should preserve the proportions of the classes (`TRUE`, the default), for `protocol = "crossvalidation"` and for `protocol = "holdout"` when it draws its own split. Ignored for a numeric target, and for the bootstrap and leave-one-out protocols, which have no split to stratify.

`...` 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)
# The simplest use: a training set, a test set, and the score of the model fitted on the
# first and evaluated on the second. Same thing as
# evaluation.accuracy (predict (NB (d$train.x, d$train.y), d$test.x), d$test.y).
d = splitdata (iris, 5, seed = 0)
performance (NB, d$train.x, d$train.y, d$test.x, d$test.y)
# Several methods and criteria at once
performance (c (NB, LDA, CART), d$train.x, d$train.y, d$test.x, d$test.y,
             eval = c ("accuracy", "kappa"))
# 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
data (linsep)
performance (c (NB, LDA, LR), linsep [, -3], linsep [, 3], type = "roc",
             protocol = "loocv", seed = 0)
# Same curves, read from the hard predicted labels instead of the class-membership
# scores: each method collapses to a single operating point.
performance (c (NB, LDA, LR), linsep [, -3], linsep [, 3], type = "roc",
             protocol = "loocv", seed = 0, fuzzy = FALSE)
# Choosing the positive class explicitly
performance (NB, linsep [, -3], linsep [, 3], type = "roc", protocol = "loocv",
             seed = 0, positive = levels (linsep [, 3]) [2])
# List of methods in a variable, confusion matrix, hodout estimation
classif = c (NB, LDA, LR)
performance (classif, iris [, -5], iris [, 5], type = "confusion",
             protocol = "holdout", seed = 0, names = c ("NB", "LDA", "LR"))
# List of strings (method names), scatterplot evaluation, crossvalidation estimation
classif = c ("NB", "LDA", "LR")
performance (classif, iris [, -5], iris [, 5], type = "scatter",
             protocol = "crossvalidation", seed = 0)
# Actual vs. predicted
data (trees)
performance (LINREG, trees [, -3], trees [, 3], type = "avsp")

## End(Not run)
```

---

plot.apriori *Plot function for apriori-class*

Description

Plot the association rules obtained by APRIORI, using arulesViz.

Usage

```
## S3 method for class 'apriori'
plot(
  x,
  method = "scatterplot",
  measure = c("support", "confidence"),
  shading = "lift",
  ...
)
```

Arguments

x	The classification model (object of class apriori-class, created by APRIORI).
method	The type of plot (see plot, e.g. "scatterplot", "graph", "grouped", "paracoord").
measure, shading	Parameters passed to plot.
...	Other parameters passed to plot.

See Also

APRIORI, apriori-class, plot

Examples

```
## Not run:
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)
plot (model)
plot (model, method = "graph")

## 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, legendpos = "topleft", ...)
```

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). Ignored when there is only one, which is what two classes give: the axis is then drawn against the observation index, and the class means as horizontal lines.
legendpos	Position of the legend, as in plotdata.
...	Other parameters, passed to the underlying plot.

See Also

CDA, predict.cda, cda-class

Examples

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

---

plot.factorial *Plot function for factorial-class*

Description

Plot PCA, CA or MCA.

Usage

```
## S3 method for class 'factorial'
plot(
  x,
  type = c("ind", "cor", "eig"),
  axes = c(1, 2),
  col = NULL,
  pch = NULL,
  labels = FALSE,
  legendpos = "topleft",
  ...
)
```

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).
col	Color(s) of the individuals (type = "ind"). If NULL (the default) and a qualitative supplementary variable was given, the individuals are colored by it, as plotdata does; otherwise any value the base col parameter accepts, so that an external clustering can be used.
pch	Point style(s) of the individuals on the scatter plot (type = "ind", PCA only – plot.CA and plot.MCA, which draw the CA and MCA plots, have no equivalent argument). Same default/auto-detection logic as col. Accepts the same values as the base pch graphical parameter.
labels	Whether the row names are shown instead of points (type = "ind").
legendpos	Position of the legend (type = "ind", PCA only, when individuals are colored by a qualitative variable).
...	Other parameters.

See Also

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

Examples

```
require (datasets)
data (iris)
pca = PCA (iris, quali.sup = 5)
plot (pca) # Automatically colored/legended by the qualitative supplementary variable
plot (pca, type = "cor")
plot (pca, type = "eig")
# Overriding colors and point styles manually (e.g. by an external clustering)
km = KMEANS (iris [, -5], k = 3)
plot (pca, col = km$cluster + 1, pch = km$cluster + 1)
```

---

plot.selection *Plot a feature selection*

Description

Draws the score every variable obtained, the ones that were kept apart from the ones that were not. Only a selection made by ranking the variables (algorithm = "ranking") can be drawn this way: it is the only one that scores them one by one, where the other three score whole subsets.

Usage

```
## S3 method for class 'selection'
plot(x, horiz = TRUE, legendpos = "bottomright", ...)
```

Arguments

x	The selection (object of class selection-class, created by selectfeatures).
horiz	Whether the bars are drawn horizontally, which leaves room for long variable names.
legendpos	Position of the legend.
...	Other parameters, passed to barplot.

See Also

selectfeatures, selection-class, print.selection

Examples

```
require (datasets)
data (iris)
# How useful a random forest finds each variable, and the two it would keep
selection = selectfeatures (iris [, -5], iris [, 5], unieval = "randomforest", uninb = 2)
selection
plot (selection)
```

---

plot.som *Plot function for som-class*

Description

Plot Kohonen's self-organizing maps.

Usage

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

Arguments

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

See Also

SOM, som-class

Examples

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

`plotavsp` *Plot actual vs. predictions*

Description

Plot actual vs. predictions of a regression model.

Usage

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

Arguments

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

See Also

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

Examples

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

plotcloud *Plot word cloud*

Description

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

Usage

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

Arguments

corpus	The corpus of documents (a vector of characters) or the vocabulary of the documents (result of function getvocab).
k	A categorical variable (vector or factor).
stopwords	The language whose stop words are removed ("en", ...), or NULL to keep them. A list of words of your own goes to excludewords.
...	Other parameters.

See Also

plotzipf, getvocab, wordcloud

Examples

```
data (capitals)
plotcloud (capitals)
vocab = getvocab (capitals, mincount = 1, lang = NULL, stopwords = "en")
plotcloud (vocab)
```

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], k = 3, method = "ward")
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,
  target = NULL,
 type = c("pairs", "scatter", "parallel", "boxplot", "histogram", "barplot", "pie",
  "heatmap", "heatmapc", "correlation", "pca", "cda", "svd", "nmf", "tsne", "som",
    "words"),
  legendpos = "topleft",
  alpha = 200,
  asp = 1,
  labels = FALSE,
  tsne = NULL,
  nmf = NULL,
  ...
)
```

Arguments

d	A numeric dataset (data.frame or matrix).
k	The variable the observations are told apart by: they are coloured, grouped or, for type = "cda", discriminated by it. It is categorical, and cluster numbers do just as well as a factor. Left NULL, it is taken from d when exactly one of its columns is qualitative, or from target when that one is categorical.
target	The variable to be explained, read by type = "correlation" only. Unlike k it may be continuous. Give one or the other, the way cutree takes either k or h: a categorical target also serves as k, and k serves as target when none is given. A continuous target leaves the observations uncoloured, having no groups to offer.
type	The type of graphic to be plotted. See the Details section on the projections.
legendpos	Position of the legend
alpha	Opacity of the plotted points, from 0 (invisible) to 255 (opaque). Useful on dense scatter plots, where points would otherwise hide each other. The legend stays opaque.
asp	Aspect ratio: 1 (the default) makes one unit as long on both axes, NA lets them scale independently.

labels	Indicates whether or not labels (row names) should be showned on the (scatter) plot.
tsne	A precomputed TSNE result. When type = "tsne", providing this avoids recomputing the (randomized, potentially costly) t-SNE embedding on every call; if NULL (default), it is computed internally as before.
nmf	A precomputed NMF result. When type = "nmf", providing this avoids recomputing the (randomized, potentially costly) NMF decomposition on every call; if NULL (default), it is computed internally as before.
...	Other parameters.

Details

type = "correlation" draws how strongly each variable relates to the target, sorted, strongest at the top. **Two different quantities, depending on the target.** Against a numeric target it is Pearson's correlation r, sign included, and the axis says so. Against a categorical one it is the **correlation ratio** η, the square root of the between-class share of the variance – *not* a Pearson coefficient computed on class numbers, which would depend on the order the classes happen to be in and would mean nothing beyond two classes. η lies in [0, 1], is defined for any number of classes and does not depend on their coding. On exactly two classes η is $|r|$ with the classes coded 0/1, so the sign comes back and says which class the variable is larger in.

The projections (type = "pca", and the default "scatter"/"pairs" on more than two variables) are computed on the data as they are, unscaled – plotdata shows a dataset, whereas PCA performs a factorial analysis and centres and scales by default. So plotdata (d, type = "pca") and plot (PCA (d)) differ visibly when the variables have very different scales, and PCA is the one to use for a properly scaled projection.

Examples

```
require (datasets)
data (iris)
# Without classification
plotdata (iris [, -5]) # Default (pairs)
# With classification
plotdata (iris [, -5], iris [, 5]) # Default (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]) # Default (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]) # Default (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
## Not run:
# Reusing a previously computed t-SNE embedding instead of recomputing it
res = TSNE (iris [, -5])
plotdata (iris [, -5], iris [, 5], type = "tsne", tsne = res)

## End(Not run)
```

`plotzipf` *Plot rank versus frequency*

Description

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

Usage

```
plotzipf(corpus)
```

Arguments

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

See Also

`plotcloud`, `getvocab`

Examples

```
data (capitals)
plotzipf (capitals)
vocab = getvocab (capitals, mincount = 1, lang = NULL)
plotzipf (vocab)
```

POLYREG *Polynomial Regression*

Description

This function builds a polynomial regression model.

Usage

```
POLYREG(
  x,
  y,
  degree = 2,
  tune = FALSE,
  methodparameters = NULL,
  graph = FALSE,
  seed = NULL,
  ...
)
```

Arguments

x	Predictor matrix.
y	Response vector.
degree	The polynom degree.
tune	If true, the function returns parameters instead of a classification model.
methodparameters	Present for interface consistency with performance (which always passes it when fitting a model). Currently unused: POLYREG does not support reusing pre-tuned parameters.
graph	Present for interface consistency with performance (which always passes it when fitting a model). Currently unused: POLYREG does not produce a plot.
seed	A specified seed for random number generation, so that two runs on the same data give the same model. Every learning method accepts it, so that it can be set the same way whatever the method; the deterministic ones simply have nothing to draw and give the same model with or without it.
...	Other parameters.

Value

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

See Also

polyreg

Examples

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

## End(Not run)
```

`predict.apriori` *Model predictions*

Description

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

Usage

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

Arguments

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

Value

A vector of predicted values (`factor`).

See Also

`APRIORI`, `apriori-class`, `apriori`

Examples

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

predict.boosting *Model predictions*

Description

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

Usage

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

Arguments

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

Value

A vector of predicted values (factor).

See Also

ADABOOST, BAGGING, boosting-class

Examples

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

## End(Not run)
```

predict.cda *Model predictions*

Description

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

Usage

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

Arguments

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

Value

A vector of predicted values (factor).

See Also

CDA, plot.cda, cda-class

Examples

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

predict.dbs *Predict function for DBSCAN*

Description

Return the closest DBSCAN cluster for a new dataset.

Usage

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

Arguments

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

See Also

DBSCAN

Examples

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

---

predict.em *Predict function for EM*

Description

Return the closest EM cluster for a new dataset.

Usage

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

Arguments

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

See Also

EM

Examples

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

`predict.factorial` *Projection of new observations into a factorial space*

Description

Projects new observations into the factorial space computed by `CA`, `MCA` or `PCA` – the same operation that is applied to the observations the analysis was fitted on: centering (and, for `PCA` with `scale.unit = TRUE`, scaling) with the parameters *of the training data*, then projection on the axes already computed. The axes are not recomputed and the new observations have no influence on them: they are *supplementary* individuals.

Usage

```
## S3 method for class 'factorial'
predict(object, test, ...)
```

Arguments

`object`	The factorial analysis (object of class `factorial-class`).
`test`	The new observations, a `data.frame` or `matrix` with the same (active) variables as the data the analysis was fitted on.
`...`	Other parameters.

Details

The projection is obtained by handing the new rows back to **FactoMineR** as supplementary individuals of the original analysis, so the coordinates are exactly the ones `PCA (rbind (train, test), ind.sup = ...)` would give. The active analysis is refitted in the process, which is unnoticeable on the sizes this package is meant for.

Supplementary variables (`quanti.sup`, `quali.sup`) play no part in the axes, so `test` does not have to carry them: any column of the training data that is missing from `test` is filled in (with the training mean, or the first level) purely so that the two can be stacked.

Value

The coordinates of the new observations on the factorial axes (a `matrix`, one row per observation and one column per axis).

See Also

`PCA`, `CA`, `MCA`, `factorial-class`, `predict.cda`

Examples

```
require (datasets)
data (iris)
d = splitdata (iris, 5)
pca = PCA (d$train.x)
# The coordinates of unseen observations on the axes of the training analysis
head (predict (pca, d$test.x))
# An observation of the training set projects onto the coordinates the analysis gave it
pca$ind$coord [1, ]
predict (pca, d$train.x [1, ])
```

`predict.hca` *Predict function for hierarchical clustering*

Description

Returns the cluster whose centre is closest, for a new dataset. A dendrogram says nothing about observations it was not built on, so the rule is the usual one: the clusters of the cut are summarised by their centres, and a new observation joins the nearest.

Usage

```
## S3 method for class 'hca'
predict(object, newdata, k = NULL, ...)
```

Arguments

`object`	The clustering (created by `HCA`).
`newdata`	A new dataset (a `data.frame`), with the same variables as the learning dataset.
`k`	The number of clusters the dendrogram is cut into. Defaults to the cut `HCA` already made, when it made one.
`...`	Other parameters.

Value

A vector of cluster numbers.

See Also

`HCA`, `predict.kmeans`

Examples

```
require (datasets)
data (iris)
d = splitdata (iris, 5)
model = HCA (d$train.x, k = 3, method = "ward")
table (predict (model, d$test.x), d$test.y)
```

predict.kmeans *Predict function for K-means*

Description

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

Usage

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

Arguments

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

See Also

KMEANS

Examples

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

predict.knn *Model predictions*

Description

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

Usage

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

Arguments

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

Value

A vector of predicted values (factor).

See Also

KNN, knn-class

Examples

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

---

predict.meanshift *Predict function for MeanShift*

Description

Return the closest MeanShift cluster for a new dataset.

Usage

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

Arguments

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

See Also

MEANSHIFT

Examples

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

## End(Not run)
```

`predict.model` *Model predictions*

Description

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

Usage

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

Arguments

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

Value

A vector of predicted values (`factor`).

See Also

`model-class`

Examples

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

predict.pam *Predict function for PAM*

Description

Returns the cluster of the closest medoid, for a new dataset.

Usage

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

Arguments

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

Value

A vector of cluster numbers.

See Also

PAM, predict.kmeans

Examples

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

predict.selection *Model predictions*

Description

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

Usage

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

Arguments

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

Value

A vector of predicted values (factor).

See Also

FEATURESELECTION, selection-class

Examples

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

## End(Not run)
```

---

predict.som *Predict function for a self-organising map*

Description

Returns the cluster of the closest unit of the map, for a new dataset.

Usage

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

Arguments

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

Value

A vector of cluster numbers.

See Also

SOM, plot.som

Examples

```
require (datasets)
data (iris)
d = splitdata (iris, 5)
model = SOM (d$train.x, 4, 4)
table (predict (model, d$test.x), d$test.y)
```

---

predict.spectral *Predict function for Spectral clustering*

Description

Return the closest Spectral clustering cluster for a new dataset. New instances are assigned to the cluster of their nearest neighbour in the original (training) space, since the spectral projection cannot be directly extended to unseen data without recomputing the affinity matrix.

Usage

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

Arguments

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

See Also

SPECTRAL, spectral-class

Examples

```
## Not run:
require (datasets)
data (iris)
d = splitdata (iris, 5)
model = SPECTRAL (d$train.x, k = 3)
predict (model, d$test.x)

## End(Not run)
```

predict.textmining *Model predictions*

Description

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

Usage

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

Arguments

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

Value

A vector of predicted values (factor).

See Also

TEXTMINING, textmining-class

Examples

```
require (text2vec)
# A small subset of movie_review is used here so this example runs quickly.
data ("movie_review")
d = movie_review [1:300, 2:3]
d [, 1] = factor (d [, 1])
d = splitdata (d, 1)
model = TEXTMINING (d$train.x, NB, labels = d$train.y, mincount = 10)
pred = predict (model, d$test.x)
evaluation (pred, d$test.y)
```

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.boosting *Print an ensemble model*

Description

Prints how many models the ensemble holds and what it was fitted on.

Usage

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

Arguments

x	The model (object of class boosting-class, created by ADABOOST or BAGGING).
...	Other parameters.

See Also

boosting-class, ADABOOST, BAGGING

Examples

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

---

print.cda *Print a canonical discriminant analysis*

Description

Prints the size of the analysis and the variance carried by its axes.

Usage

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

Arguments

x	The model (object of class cda-class, created by CDA).
...	Other parameters.

See Also

cda-class, CDA, plot.cda

Examples

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

print.dataset *Print a training/test split*

Description

Prints the sizes of the two parts of a split and the target they share.

Usage

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

Arguments

x	The split (object of class dataset-class, created by splitdata).
...	Other parameters.

See Also

dataset-class, splitdata

Examples

```
require (datasets)
data (iris)
splitdata (iris, 5)
```

print.dbs *Print a DBSCAN clustering*

Description

Prints the number of clusters found, their sizes, the observations left as noise, and the two parameters used – instead of dumping the underlying list.

Usage

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

Arguments

x	The clustering (object of class dbs-class, created by DBSCAN).
...	Other parameters.

See Also

dbs-class, DBSCAN

Examples

```
require (datasets)
data (iris)
DBSCAN (iris [, -5], minpts = 5, eps = 0.65)
```

---

print.em *Print an EM clustering*

Description

Prints the number of clusters found, their sizes and the log-likelihood reached.

Usage

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

Arguments

x	The clustering (object of class em-class, created by EM).
...	Other parameters.

See Also

em-class, EM

Examples

```
require (datasets)
data (iris)
EM (iris [, -5], 3)
```

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)
```

print.knn *Print a* K*-nearest-neighbours model*

Description

Prints the size of the training set the model memorised, and the number of neighbours used.

Usage

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

Arguments

x	The model (object of class knn-class, created by KNN).
...	Other parameters.

See Also

knn-class, KNN

Examples

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

---

print.meanshift *Print a mean shift clustering*

Description

Prints the number of clusters found, their sizes and the kernel used.

Usage

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

Arguments

x	The clustering (object of class meanshift-class, created by MEANSHIFT).
...	Other parameters.

See Also

meanshift-class, MEANSHIFT

Examples

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

print.model *Print a classification or regression model*

Description

Prints a short description of the model – the method it was obtained with, and the size of the data it was fitted on – instead of dumping the underlying list. Use `summary(model)` for the full detail of the wrapped model.

Usage

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

Arguments

x	The model to be printed (object of class model-class).
...	Other parameters.

See Also

model-class, summary.model, predict.model

Examples

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

print.params *Print tuned method parameters*

Description

Prints the hyperparameters a method retained, or says that it has none.

Usage

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

Arguments

x	The parameters (object of class params-class, obtained by calling a classification method with tune = TRUE).
...	Other parameters.

See Also

params-class, performance

Examples

```
require (datasets)
data (iris)
# A small grid, so that the example stays fast; the defaults search a much larger one.
SVM (iris [, -5], iris [, 5], gamma = 2^(-2:0), cost = 2^(0:2), tune = TRUE)
# A method with nothing to tune says so.
NB (iris [, -5], iris [, 5], tune = TRUE)
```

---

print.selection *Print a feature selection result*

Description

Prints which features were selected, by which algorithm and criteria, instead of dumping the underlying list.

Usage

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

Arguments

x	The result (object of class selection-class, created by selectfeatures).
...	Other parameters.

See Also

selection-class, selectfeatures

Examples

```
require (datasets)
data (iris)
selectfeatures (iris [, -5], iris [, 5], algorithm = "forward", multieval = "cfs")
```

print.som *Print a self-organising map*

Description

Prints the size of the map, how many of its units are actually used, and the dataset it was fitted on.

Usage

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

Arguments

x	The map (object of class som-class, created by SOM).
...	Other parameters.

See Also

som-class, SOM

Examples

```
require (datasets)
data (iris)
SOM (iris [, -5], 4, 4)
```

print.spectral *Print a spectral clustering*

Description

Prints the number of clusters found, their sizes and the dimension of the spectral projection.

Usage

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

Arguments

x	The clustering (object of class spectral-class, created by SPECTRAL).
...	Other parameters.

See Also

spectral-class, SPECTRAL

Examples

```
require (datasets)
data (iris)
SPECTRAL (iris [, -5], 3)
```

---

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,
  methodparameters = NULL,
  graph = FALSE,
  seed = NULL,
  ...
)
```

Arguments

train	The training set (description), as a data.frame.
labels	Class labels of the training set (vector or factor).
tune	If true, the function returns parameters instead of a classification model.
methodparameters	Present for interface consistency with performance (which always passes it when fitting a model). Currently unused: QDA does not support reusing pre-tuned parameters.
graph	Present for interface consistency with performance (which always passes it when fitting a model). Currently unused: QDA does not produce a plot.
seed	A specified seed for random number generation, so that two runs on the same data give the same model. Every learning method accepts it, so that it can be set the same way whatever the method; the deterministic ones simply have nothing to draw and give the same model with or without it.
...	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 that 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

```
require (text2vec)
# A small subset of movie_review is used here so this example runs quickly.
data (movie_review)
reviews = movie_review$review [1:300]
vectorizer = vectorize.docs (corpus = reviews, returndata = FALSE)
docs = vectorize.docs (corpus = reviews, vectorizer = vectorizer)
query.docs (docs, reviews [1], vectorizer)
query.docs (docs, docs [1, ], vectorizer)
```

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

```
# 'capitals' is small, so the word vectors are coarse and 'ndim' is reduced
# accordingly; phrase detection needs a much larger corpus.
data (capitals)
words = vectorize.words (capitals, mincount = 2, ndim = 10, maxiter = 5)
query.words (words, origin = "paris", sub = "france", add = "germany")
query.words (words, origin = "berlin", sub = "germany", add = "france")
```

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,
  methodparameters = NULL,
  graph = FALSE,
  seed = NULL,
  ...
)
```

Arguments

train	The training set (description), as a data.frame.
labels	Class labels of the training set (vector or factor).
ntree	The number of trees in the forest.
nvar	Number of variables randomly sampled as candidates at each split.
tune	If true, the function returns parameters instead of a classification model.
methodparameters	Present for interface consistency with performance (which always passes it when fitting a model). Currently unused: RANDOMFOREST does not yet implement hyperparameter tuning, so tune = TRUE returns an empty params object and there is nothing for methodparameters to override.
graph	Present for interface consistency with performance (which always passes it when fitting a model). Currently unused: RANDOMFOREST does not produce a plot.
seed	A specified seed for random number generation (bootstrap sampling of the trees and, when nvar is smaller than the total number of variables, the candidate variables drawn at each split).
...	Other parameters, forwarded to randomForest.

Value

The classification model.

See Also

randomForest

Examples

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

---

reg1 *reg1 dataset*

Description

Artificial dataset for simple regression tasks.

Usage

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

Format

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

Author(s)

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

---

reg2 *reg2 dataset*

Description

Artificial dataset for simple regression tasks.

Usage

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

Format

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

Author(s)

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

---

regplot *Plot function for a regression model*

Description

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

Description

Plot the studentized residuals of a linear regression model.

Usage

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

Arguments

model	The model to be plotted.
index	The index of the variable used for 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" (dependent variable)
resplot (model, index = 1) # Ordered by variable "Girth" (independent variable)
resplot (model, index = 2) # Ordered by variable "Height" (independent variable)
```

---

roc.curves *Plot ROC Curves*

Description

This function plots ROC Curves of one or several classification predictions.

Usage

```
roc.curves(
  predictions,
  gt,
  methods.names = NULL,
  positive = levels(factor(gt))[1],
  type = c("auto", "fuzzy", "hard"),
  ...
)
```

Arguments

predictions	The predictions of one or several classification models. Four shapes are accepted: a `factor` of hard labels (one model); a numeric `vector` of scores for the positive class (one model); a `matrix` of class probabilities, i.e. one column per class named after it, as returned by `predict (model, x, fuzzy = TRUE)` (one model); or any other `matrix`/`data.frame`, read as one column per model.
gt	Actual labels of the dataset (`factor` or `vector`), two classes only.
methods.names	The name of the compared methods (`vector`).
positive	The label of the positive class. Defaults to the first level of `gt`, as everywhere else in the package – note that for the usual alphabetical ordering this is often the *negative* class, so it is worth setting explicitly.
type	`"auto"` (default) reads `predictions` according to its shape, `"fuzzy"` requires scores and refuses hard labels, `"hard"` reduces everything to the predicted class first (this is the coarse three-point curve discussed above).
...	Other parameters, passed to the underlying plot.

Details

A ROC curve needs a *score*: the higher it is, the more likely the observation is to belong to the positive class. The natural one is the estimated probability returned by `predict(model, x, fuzzy = TRUE)`. Hard class labels give only two distinct values, so the "curve" reduces to three points and the area under it says very little. That coarse version is available on purpose – it makes a useful comparison – but it has to be asked for: pass hard labels, or `type = "hard"`.

Value

Nothing; the curves are drawn on the current graphics device.

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])
# From the estimated probabilities: the meaningful curve
roc.curves (predict (model.nb, d [, -5], fuzzy = TRUE), d [, 5], positive = "+")
# Two models compared, one score column each
roc.curves (cbind (NB = predict (model.nb, d [, -5], fuzzy = TRUE) [, "+"],
                   LDA = predict (model.lda, d [, -5], fuzzy = TRUE) [, "+"]),
            d [, 5], c ("NB", "LDA"), positive = "+")
# The same predictions reduced to hard labels: three points, and little to read
roc.curves (predict (model.nb, d [, -5]), d [, 5], positive = "+", type = "hard")
```

---

rotation *Rotation*

Description

Rotation on two variables of a numeric dataset

Usage

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

Arguments

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

Value

A rotated data matrix.

Examples

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

`runningtime` *Running time*

Description

Return the running time of a function

Usage

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

Arguments

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

Value

The running time of function `FUN`.

See Also

`difftime`

Examples

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

`scatterplot` *Clustering Scatter Plots*

Description

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

Usage

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

Arguments

`d`	The dataset (`matrix` or `data.frame`).
`clusters`	Cluster labels of the training set: a numeric `vector` (0 marking the observations a density method left as noise), or a `factor`/character vector, whose levels are then used to label the legend.
`centers`	Coordinates of the cluster centers.
`labels`	Indicates whether or not labels (row names) should be showned on the plot.
`ellipses`	Indicates whether or not ellipses should be drawned around clusters.
`legend`	Indicates where the legend is placed on the graphics.
`...`	Other parameters.

Examples

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

`selectfeatures` *Feature selection for classification*

Description

Select a subset of features for a classification task.

Usage

```
selectfeatures(
  train,
  labels,
  algorithm = c("ranking", "forward", "backward", "exhaustive"),
  unieval = if (algorithm[1] == "ranking") fseval.univariate() else NULL,
  uninb = NULL,
  unithreshold = NULL,
  multieval = fseval.multivariate(),
  wrapmethod = NULL,
  keep = FALSE,
  ...
)
```

Arguments

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

See Also

`FEATURESELECTION`, `selection-class`

Examples

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

## End(Not run)
```

---

selection-class *Feature selection*

Description

This class contains the result of feature selection algorithms.

Details

Objects of this class are plain lists with the following components:

`selection` A vector of integers indicating the selected features.

`features` The names of the selected features, when the dataset had column names.

`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

Originally published by G. Hunault, Departement Informatique, Universite d'Angers, as the "RONFLE" file of his statistics dataset collection. That collection has changed address twice and its current host was unreachable when this version was prepared, so the citation is to an archived copy: https://web.archive.org/web/20250319122109/https://gilles-hunault.leria-info.univ-angers.fr/Datasets/datasets.htm.

SOM *Self-Organizing Maps clustering method*

Description

Run the SOM algorithm for clustering.

Usage

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

Arguments

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

Details

Objects of this class are plain lists with the following components:

`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 = FALSE, seed = NULL, ...)
```

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).
`seed`	A specified seed for random number generation (final k-means step).
`...`	Other parameters.

See Also

`spectral-class`

Examples

```
require (datasets)
data (iris)
SPECTRAL (iris [, -5], k = 3)
```

spectral-class *Spectral clustering model*

Description

This class contains the model obtained by Spectral clustering.

Details

Objects of this class are plain lists with the following components:

`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).

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

See Also

`SPECTRAL`, `predict.spectral`

spine *Spine dataset*

Description

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

Usage

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

Format

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

Source

https://archive.ics.uci.edu/dataset/212/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,
  stratify = TRUE
)
```

Arguments

dataset	The dataset to be split (data.frame or matrix).
target	The column index (numeric) or column name (character) of the target variable (class label or response variable).
size	The size of the training set: either a number of observations, or a proportion between 0 and 1.
seed	A specified seed for random number generation.
stratify	Whether the split preserves the proportions of the classes. It matters as soon as they are imbalanced: a plain random split can leave a rare class out of one side altogether. Ignored when the target is numeric.

Value

An object of class dataset-class.

See Also

dataset-class

Examples

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

`stability` *Clustering evaluation through stability*

Description

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

Usage

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

Arguments

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

Value

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

See Also

`compare`, `intern`

Examples

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

## End(Not run)
```

STUMP *Classification using one-level decision tree*

Description

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

Usage

```
STUMP(
  train,
  labels,
  randomvar = FALSE,
  tune = FALSE,
  methodparameters = NULL,
  graph = FALSE,
  seed = NULL,
  ...
)
```

Arguments

train	The training set (description), as a data.frame.
labels	Class labels of the training set (vector or factor).
randomvar	If TRUE, the stump is built on a single variable drawn at random instead of the best one (useful to build weak learners for an ensemble method). Note that the model then differs from one call to the next unless seed is set. Defaults to FALSE, i.e. the usual decision stump, split on the variable selected by CART.

tune	If true, the function returns parameters instead of a classification model.
methodparameters	Present for interface consistency with performance (which always passes it when fitting a model). Currently unused: STUMP does not support reusing pre-tuned parameters.
graph	Present for interface consistency with performance (which always passes it when fitting a model). Currently unused: STUMP does not produce a plot.
seed	A specified seed for random number generation (used only if randomvar is TRUE).
...	Other parameters.

Value

The classification model.

See Also

CART

Examples

```
require (datasets)
data (iris)
STUMP (iris [, -5], iris [, 5])
STUMP (iris [, -5], iris [, 5], randomvar = TRUE, seed = 0)
```

---

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

Description

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

Usage

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

Arguments

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

See Also

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

Examples

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

---

summary.model *Summary of a classification or regression model*

Description

Prints the short description of print.model, followed by the summary of the wrapped model itself.

Usage

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

Arguments

object	The model (object of class model-class).
...	Other parameters, passed to the summary of the wrapped model.

See Also

model-class, print.model

Examples

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

SVD *Singular Value Decomposition*

Description

Return the SVD decomposition.

Usage

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

Arguments

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

See Also

svd

Examples

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

SVM *Classification using Support Vector Machine*

Description

This function builds a classification model using Support Vector Machine.

Usage

```
SVM(
  train,
  labels,
  gamma = 2^(-3:3),
  cost = 2^(-3:3),
  kernel = c("radial", "linear"),
  nfolds = 10,
  tune = FALSE,
  methodparameters = NULL,
  graph = FALSE,
  seed = NULL,
  ...
)
```

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.
nfolds	The number of folds of the cross-validation a method runs to choose its hyperparameters. Only used when there is something to choose, i.e. when one of them is given as a vector. Lower it to fit faster, at the cost of a noisier choice.
tune	If true, the function returns parameters instead of a classification model.
methodparameters	Object containing the parameters. If given, it replaces gamma and cost.
graph	Present for interface consistency with performance (which always passes it when fitting a model). Currently unused: SVM does not produce a plot.
seed	A specified seed for random number generation, so that two runs on the same data give the same model. Every learning method accepts it, so that it can be set the same way whatever the method; the deterministic ones simply have nothing to draw and give the same model with or without it.
...	Other arguments.

Value

The classification model.

See Also

svm, SVMl, SVMr

Examples

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

## End(Not run)
```

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

Description

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

Usage

```
SVMl(
  train,
  labels,
  cost = 2^(-3:3),
  nfolds = 10,
  tune = FALSE,
  methodparameters = NULL,
  graph = FALSE,
  seed = NULL,
  ...
)
```

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).
nfolds	The number of folds of the cross-validation a method runs to choose its hyperparameters. Only used when there is something to choose, i.e. when one of them is given as a vector. Lower it to fit faster, at the cost of a noisier choice.
tune	If true, the function returns parameters instead of a classification model.
methodparameters	Object containing the parameters. If given, it replaces gamma and cost.
graph	Whether the method draws the graphic that goes with its tuning (the cross-validation curve, typically). Methods that have no such graphic accept the argument and ignore it.
seed	A specified seed for random number generation, so that two runs on the same data give the same model. Every learning method accepts it, so that it can be set the same way whatever the method; the deterministic ones simply have nothing to draw and give the same model with or without it.
...	Other arguments.

Value

The classification model.

See Also

svm, SVM

Examples

```
## Not run:
require (datasets)
data (iris)
```

```
SVMl (iris [, -5], iris [, 5], cost = 1)

## End(Not run)
```

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

Description

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

Usage

```
SVMr(
  train,
  labels,
  gamma = 2^(-3:3),
  cost = 2^(-3:3),
  nfolds = 10,
  tune = FALSE,
  methodparameters = NULL,
  graph = FALSE,
  seed = NULL,
  ...
)
```

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).
nfolds	The number of folds of the cross-validation a method runs to choose its hyperparameters. Only used when there is something to choose, i.e. when one of them is given as a vector. Lower it to fit faster, at the cost of a noisier choice.
tune	If true, the function returns parameters instead of a classification model.
methodparameters	Object containing the parameters. If given, it replaces gamma and cost.
graph	Whether the method draws the graphic that goes with its tuning (the cross-validation curve, typically). Methods that have no such graphic accept the argument and ignore it.

seed	A specified seed for random number generation, so that two runs on the same data give the same model. Every learning method accepts it, so that it can be set the same way whatever the method; the deterministic ones simply have nothing to draw and give the same model with or without it.
...	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),
  nfolds = 10,
  tune = FALSE,
  methodparameters = NULL,
  graph = FALSE,
  seed = NULL,
  ...
)
```

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).
nfolds	The number of folds of the cross-validation a method runs to choose its hyperparameters. Only used when there is something to choose, i.e. when one of them is given as a vector. Lower it to fit faster, at the cost of a noisier choice.
tune	If true, the function returns parameters instead of a classification model.
methodparameters	Object containing the parameters. If given, it replaces epsilon, gamma and cost. Named (and behaves identically to) methodparameters rather than params, for consistency with SVM and with the calling convention used by performance/the internal protocol.* functions, which always pass a methodparameters argument.
graph	Present for interface consistency with performance (which always passes it when fitting a model). Currently unused: SVR does not produce a plot.
seed	A specified seed for random number generation, so that two runs on the same data give the same model. Every learning method accepts it, so that it can be set the same way whatever the method; the deterministic ones simply have nothing to draw and give the same model with or without it.
...	Other arguments.

Value

The classification model.

See Also

svm, SVRl, SVRr

Examples

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

## End(Not run)
```

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

Description

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

Usage

```
SVRl(
  x,
  y,
  cost = 2^(-3:3),
  epsilon = c(0.1, 0.5, 1),
  nfolds = 10,
  tune = FALSE,
  methodparameters = NULL,
  graph = FALSE,
  seed = NULL,
  ...
)
```

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).
nfolds	The number of folds of the cross-validation a method runs to choose its hyperparameters. Only used when there is something to choose, i.e. when one of them is given as a vector. Lower it to fit faster, at the cost of a noisier choice.
tune	If true, the function returns parameters instead of a classification model.
methodparameters	Object containing the parameters. If given, it replaces epsilon, gamma and cost. Named to match SVR (see there).
graph	Whether the method draws the graphic that goes with its tuning (the cross-validation curve, typically). Methods that have no such graphic accept the argument and ignore it.
seed	A specified seed for random number generation, so that two runs on the same data give the same model. Every learning method accepts it, so that it can be set the same way whatever the method; the deterministic ones simply have nothing to draw and give the same model with or without it.
...	Other arguments.

Value

The classification model.

See Also

svm, SVR

Examples

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

## End(Not run)
```

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

Description

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

Usage

```
SVRr(
  x,
  y,
  gamma = 2^(-3:3),
  cost = 2^(-3:3),
  epsilon = c(0.1, 0.5, 1),
  nfolds = 10,
  tune = FALSE,
  methodparameters = NULL,
  graph = FALSE,
  seed = NULL,
  ...
)
```

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).
nfolds	The number of folds of the cross-validation a method runs to choose its hyperparameters. Only used when there is something to choose, i.e. when one of them is given as a vector. Lower it to fit faster, at the cost of a noisier choice.
tune	If true, the function returns parameters instead of a classification model.
methodparameters	Object containing the parameters. If given, it replaces epsilon, gamma and cost. Named to match SVR (see there).
graph	Whether the method draws the graphic that goes with its tuning (the cross-validation curve, typically). Methods that have no such graphic accept the argument and ignore it.
seed	A specified seed for random number generation, so that two runs on the same data give the same model. Every learning method accepts it, so that it can be set the same way whatever the method; the deterministic ones simply have nothing to draw and give the same model with or without it.
...	Other arguments.

Value

The classification model.

See Also

svm, SVR

Examples

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

## End(Not run)
```

---

temperature *Temperature dataset*

Description

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

Usage

```
temperature
```

Format

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

TEXTMINING *Text mining*

Description

Apply data mining function on vectorized text

Usage

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

Arguments

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

Value

The result of the data mining method.

See Also

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

Examples

```
require (text2vec)
# A small subset of movie_review is used here so this example runs quickly.
data ("movie_review")
d = movie_review [1:300, 2:3]
d [, 1] = factor (d [, 1])
d = splitdata (d, 1)
model = TEXTMINING (d$train.x, NB, labels = d$train.y, mincount = 10)
pred = predict (model, d$test.x)
evaluation (pred, d$test.y)
data (capitals)
clusters = TEXTMINING (capitals, HCA, vector = "words", k = 5, mincount = 2, ndim = 10, maxiter = 5)
plotclus (clusters$res, capitals, type = "tree", labels = TRUE)
```

textmining-class *Text mining object*

Description

Object used for text mining.

Details

Objects of this class are plain lists with the following components:

`vectorizer` The vectorizer.

`vectors` The vectorized dataset.

`res` The result of the text mining method.

See Also

TEXTMINING, vectorize.docs

titanic *Titanic dataset*

Description

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

Usage

```
titanic
```

Format

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

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

`Age` Adult or Child.

`Sex` Female or Male.

`Fate` Casualty or Survivor.

Source

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

See Also

Titanic

`treeplot` *Dendrogram Plots*

Description

Draws a dendrogram.

Usage

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

Arguments

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

See Also

`dendrogram`, `HCA`, `hclust`, `agnes`

Examples

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

TSNE *t-distributed Stochastic Neighbor Embedding*

Description

Return the t-SNE dimensionality reduction.

Usage

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

Arguments

x	A numeric dataset (data.frame or matrix).
perplexity	Specification of the perplexity.
nstart	How many random sets should be chosen? The embedding with the lowest final cost is kept.
seed	A specified seed for random number generation.
...	Other parameters.

Value

The Rtsne result. Rtsne requires distinct observations, so duplicated rows of x are removed before fitting; the returned Y (and costs) are then expanded back to nrow (x) rows, in the order of x, so that they can be used directly alongside the original dataset – duplicated observations simply share the same coordinates.

See Also

Rtsne

Examples

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

universite *University dataset*

Description

The dataset presents a french university demographics.

Usage

```
universite
```

Format

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

Source

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

vectorize.docs *Document vectorization*

Description

Vectorize a corpus of documents.

Usage

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

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	The language whose stop words are removed ("en", ...), or NULL to keep them. A list of words of your own goes to excludewords.
excludewords	An optional custom vector of additional words to exclude from the vocabulary (e.g. corpus-specific stop words), on top of (or instead of) the language stopwords given through stopwords.
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.
removesinglechars	Whether single-character tokens are removed during cleanup.
sparse	Whether the document-term matrix is returned as a sparse matrix (dgCMatrix) rather than as an ordinary data.frame. A document-term matrix is mostly zeros, and storing them all takes about 4.5 GB for 20000 documents and 30000 terms, so anything but a small corpus needs TRUE. Every method of the package accepts either.
...	Other parameters.

Value

The vectorized documents, as a data.frame or, if sparse is TRUE, as a sparse matrix.

See Also

query.docs, stopwords, vectorizers

Examples

```
require (text2vec)
# A small subset of movie_review is used here so this example runs quickly.
data ("movie_review")
reviews = movie_review [1:300, ]
# Clustering
docs = vectorize.docs (corpus = reviews$review, transform = "tfidf")
km = KMEANS (docs [sample (nrow (docs), 50), ], k = 10)
# Classification
d = reviews [, 2:3]
d [, 1] = factor (d [, 1])
d = splitdata (d, 1)
vectorizer = vectorize.docs (corpus = d$train.x,
```

```
                              returndata = FALSE, mincount = 10)
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)
```

`vectorize.words` *Word vectorization*

Description

Vectorize words from a corpus of documents.

Usage

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

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`	The language whose stop words are removed (`"en"`, ...), or `NULL` to keep them. A list of words of your own goes to `excludewords`.
`excludewords`	An optional custom vector of additional words to exclude from the vocabulary (e.g. corpus-specific stop words), on top of (or instead of) the language stopwords given through `stopwords`.
`removesinglechars`	Whether single-character tokens are removed during cleanup.
`...`	Other parameters.

Value

The vectorized words.

See Also

`query.words`, `stopwords`, `vectorizers`

Examples

```
# 'capitals' is small, so the word vectors are coarse and 'ndim' is reduced
# accordingly; phrase detection needs a much larger corpus.
data (capitals)
words = vectorize.words (capitals, mincount = 2, ndim = 10, maxiter = 5)
query.words (words, origin = "paris", sub = "france", add = "germany")
query.words (words, origin = "berlin", sub = "germany", add = "france")
```

`vectorizer-class` *Document vectorization object*

Description

This class contains a vectorization model for textual documents.

Details

Objects of this class are plain lists with the following components:

`vectorizer` The vectorizer.

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

`phrases` The phrase detection method.

`tfidf` The TF-IDF transformation.

`lsa` The LSA transformation.

`tokens` The token from the original document.

See Also

vectorize.docs, query.docs

---

vowels *Vowels dataset*

Description

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

Usage

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

Format

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

Source

https://archive.ics.uci.edu/dataset/59/letter+recognition

---

wheat *Wheat dataset*

Description

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

Usage

```
wheat
```

Format

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

Source

https://archive.ics.uci.edu/dataset/236/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/dataset/109/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/dataset/111/zoo

Index

accident2014, 6
ADABOOST, 6, *12, 13*, *114*, *126, 127*
agnes, *70*, *168*
alcohol, 7
APRIORI, 8, *9*, *102*, *113*, *126*, *155*
apriori, *9*, *63*, *66*, *113*, *126*, *155*
apriori-class, 9
augmentation, 10
autompg, 10

BAGGING, *7*, 11, *13*, *114*, *126, 127*
barplot, *105*
beetles, 12
birth, 13
boosting-class, 13
boxclus, 14, *108*
boxplot, *14*
britpop, 14

CA, *15*, 15, *61*, *87*, *98*, *104*, *117*, *130*
capitals, 16
CART, 16, *18–21*, *155*
cartdepth, *17*, 18, *19–21*
cartinfo, *17, 18*, 18, *19–21*
cartleafs, *17–19*, 19, *20, 21*
cartnodes, *17–19*, 20, *21*
cartplot, *17–20*, 20
CDA, 21, *23*, *103*, *115*, *121*, *123*, *127*
cda-class, 22
closegraphics, 23, *59, 60*
clusGap, *78*
compare, 23, *25–27*, *70*, *153*
compare.accuracy, *24*, 24, *26, 27*
compare.jaccard, *24, 25*, 25, *26, 27*
compare.kappa, *24–26*, 26, *27*
confusion, 27, *46*, *101*, *106*
cookies, 28
cookplot, 29
cor, *30*
correlated, 30
cost.curves, 30, *101*, *143*
create_vocabulary, *67*
credit, 31
cutree, *109*
cv.glmnet, *82*

data.diag, 32, *33–38*
data.gauss, 33, *38*
data.parabol, *32, 33*, 34, *35–38*
data.target1, *32*, *34*, 35, *36, 37*
data.target2, *32–35*, 36, *37, 38*
data.twomoons, *32–36*, 37, *38*
data.xor, *32–37*, 38
data1, 39
data2, 39
data3, 40
dataset-class, 40
dbs-class, 41
DBSCAN, *41*, 41, *43*, *116*, *128, 129*
dbscan, *42, 43*
decathlon, 42
dendrogram, *168*
dev.off, *23*
Devices, *59*
difftime, *144*
distplot, *41, 42*, 43
download.file, *85*

EM, 43, *45*, *116*, *129*
em, *44*
em-class, 44
eucalyptus, 45
evaluation, *28*, 45, *47–50*, *52–58*, *100, 101*
evaluation.accuracy, *28*, *46*, 46, *49, 50*, *52–54*, *56*, *58*, *106*
evaluation.adjr2, 47
evaluation.fmeasure, *46, 47*, 48, *50*, *52–54*, *56*, *58*, *106*
evaluation.fowlkesmallows, *46, 47*, *49*, 49, *52–54*, *56*, *58*, *106*

evaluation.goodness, *46, 47*, *49, 50*, 51, *53, 54*, *56*, *58*, *106*
evaluation.jaccard, *46, 47*, *49, 50*, *52*, 52, *54*, *56*, *58*, *106*
evaluation.kappa, *46, 47*, *49, 50*, *52–54*, 54, *56*, *58*, *106*
evaluation.msep, *46*, *48*, 55, *57*, *106*
evaluation.precision, *28*, *46, 47*, *49, 50*, *52–54*, 55, *58*, *106*
evaluation.r2, *46*, *48*, *55*, 57, *106*
evaluation.recall, *46, 47*, *49, 50*, *52–54*, *56*, 57, *106*
exportgraphics, *23*, 59, *60*
exportgraphics.off, 60
exportgraphics.on *(*exportgraphics.off*)*, 60

factorial-class, 61
FEATURESELECTION, 61, *123*, *146, 147*
file.choose, *84*
filter.rules, 63
frequentwords, 64

GBREG, 65
general.rules, 66
getvocab, *16*, *64*, 67, *107*, *111*
glmnet, *83*, *86*
GRADIENTBOOSTING, *65, 66*, 68

HCA, 69, *118*, *168*
hclust, *70*, *168*

intern, *24*, 70, *71–73*, *153*
intern.dunn, *70*, 71, *72, 73*
intern.interclass, *70, 71*, 72, *73*
intern.intraclass, *70–72*, 72
ionosphere, 73

jpeg, *59*

kaiser, 74, *98*
KERREG, 74
KMEANS, 75, *78*, *96*, *119*, *135*
kmeans, *76–78*, *135*
kmeans.getk, 77, *96*, *135*
KNN, 78, *79*, *119, 120*, *130, 131*
knn, *79*, *120*
knn-class, 79

labc, *29*
labp, *29*
LDA, *22*, 80
lda, *22*, *80*
leverageplot, 81
LINREG, *66*, 81, *86*
linsep, 83
lm, *83*
loadtext, *16*, 84
LR, 85

MCA, *15*, *61*, *87*, 87, *98*, *104*, *117*, *130*
mclustModelNames, *44, 45*
MEANSHIFT, 88, *89*, *120*, *131*
meanShift, *89*
meanshift-class, 89
MLP, 90, *92*, *97*
MLPREG, 91, *97*
model-class, 92
movies, 93
mstep, *44*
multinom, *86*
mvr, *83*

naiveBayes, *94*
NB, 93
nirc, *29*
nirp, *29*
NMF, 94, *110*
nmf, *95*
nnet, *91, 92*
npregress, *75*

ozone, 95

PAM, 96, *122*
pam, *96*
params-class, 97
PCA, *15*, *61*, *74*, *87*, 97, *98*, *104*, *110*, *117*, *130*
performance, *7, 8*, *11, 12*, *17*, *21*, *28*, *31*, *46*, *49, 50*, *52, 53*, *56*, *58*, *62*, *65*, *68*, *75*, *79, 80*, *82*, *86*, *90*, *92*, *94*, 99, *106*, *112*, *133*, *136*, *139*, *143*, *155*, *158*, *162*
plot, *102*
plot.apriori, 101
plot.CA, *104*
plot.cda, *22, 23*, 102, *115*, *127*
plot.factorial, *15*, *61*, *87*, *98*, 103
plot.MCA, *104*

plot.PCA, *104*
plot.selection, 104
plot.som, 105, *108*, *124*, *149, 150*
plotavsp, 106
plotcloud, 107, *111*
plotclus, 107
plotdata, *103, 104*, 109
plotzipf, *67*, *107*, 111
POLYREG, 112
polyreg, *112*
postscript, *59*
predict, *92*
predict.apriori, *9*, 113, *126*, *155*
predict.boosting, *7*, *12, 13*, 114
predict.cda, *22, 23*, *103*, 115, *117*
predict.dbs, *42*, 115
predict.em, 116
predict.factorial, 117
predict.hca, *70*, 118
predict.kmeans, *76*, *118*, 119, *122*
predict.knn, *79*, 119
predict.meanshift, *89*, 120
predict.model, *92*, 121, *132*
predict.pam, 122
predict.selection, *62*, 122, *147*
predict.som, 123
predict.spectral, 124, *151*
predict.textmining, 125, *166*
print.apriori, *9*, 126, *155*
print.boosting, *13*, 126
print.CA, *130*
print.cda, 127
print.dataset, 128
print.dbs, 128
print.em, 129
print.factorial, 130
print.knn, 130
print.MCA, *130*
print.meanshift, 131
print.model, 132, *156*
print.params, 132
print.PCA, *130*
print.selection, *105*, 133
print.som, 134
print.spectral, 134
pseudoF, *78*, 135

QDA, 136
qda, *136*

query.docs, 137, *171*, *174*
query.words, 138, *173*

RANDOMFOREST, 139
randomForest, *139, 140*
reg1, 140
reg2, 140
regplot, 141
regsubsets, *83*
resplot, 141
roc.curves, *31*, *100, 101*, 142
rotation, 143
rpart, *17*
Rtsne, *169*
runningtime, 144

scatterplot, *108*, 145
selectfeatures, *62*, *105*, *133*, 146, *147*
selection-class, 147
silhouette, *78*
sim2, *137, 138*
snore, 148
SOM, *105*, *123, 124*, *134*, 148, *150*
som, *149, 150*
som-class, 149
SPECTRAL, *124*, *134, 135*, 150, *151*
spectral-class, 151
spine, 151
splitdata, *28*, *40*, *128*, 152
stability, *24*, *70*, 153
stopwords, *67*, *171*, *173*
STUMP, 154
subset, *63*
summary.apriori, *9*, *126*, 155
summary.model, *132*, 156
SVD, 156
svd, *157*
SVM, *97*, 157, *159*, *161, 162*
svm, *158, 159*, *161, 162*, *164, 165*
SVMl, *158*, 158
SVMr, *158*, 160
SVR, *66*, *97*, 161, *163–165*
SVRl, *162*, 163
SVRr, *162*, 164

temperature, 165
TEXTMINING, *125*, 166, *167*
textmining-class, 167
Titanic, *167*

titanic, 167
toggleexport, *23*, *59*
toggleexport *(*exportgraphics.off*)*, 60
treeplot, *70*, *108*, 168
TSNE, *110*, 169

universite, 170
unzip, *85*

vectorize.docs, *16*, *137*, *166, 167*, 170, *174*
vectorize.words, *16*, *138*, *166*, 172
vectorizer-class, 173
vectorizers, *171*, *173*
vowels, 174

wheat, 174
wine, 175
wordcloud, *107*

xgboost, *65, 66*, *69*

zoo, 175