

Package ‘klaR’

September 22, 2009

Version 0.6-0

Date 2009-09-11

Title Classification and visualization

Author Christian Roever, Nils Raabe, Karsten Luebke, Uwe Ligges, Gero Szepannek, Marc Zentgraf

Maintainer Uwe Ligges <ligges@statistik.tu-dortmund.de>

SystemRequirements SVMlight

Suggests scatterplot3d (>= 0.3-22), som, mlbench, rpart, e1071, combinat

Depends R (>= 2.0.0), MASS

Description Miscellaneous functions for classification and visualization developed at the Fakultät Statistik, Technische Universität Dortmund

License GPL-2

URL <http://www.statistik.tu-dortmund.de>

Repository CRAN

Date/Publication 2009-09-22 10:11:30

R topics documented:

b.scal	2
B3	4
benchB3	5
betascale	6
calc.trans	7
centerlines	8
classscatter	8
corclust	9
countries	11
dkernel	11

drawparti	12
e.scal	14
EDAM	15
errormatrix	17
friedman.data	19
greedy.wilks	20
hmm.sop	22
loclda	23
locpvs	25
meclight.default	27
NaiveBayes	29
nm	30
partimat	32
plineplot	34
plot.NaiveBayes	35
predict.loclda	36
predict.locpvs	37
predict.meclight	39
predict.NaiveBayes	40
predict.pvs	41
predict.sknn	42
predict.svmight	43
pvs	44
quadplot	47
rda	49
shardsplot	52
sknn	55
stepclass	56
svmight	58
triframe	61
trigrd	61
triperplines	63
tripplot	64
tripoints	65
tritrafo	66
ucpm	68
Index	70

b.scal

Calculation of beta scaling parameters

Description

Calculates the scaling parameter for `betascale`.

Usage

```
b.scal(member, grouping, dis = FALSE, eps = 1e-04)
```

Arguments

member	Membership values of an argmax classification method. Eg. posterior probabilities of <code>lda</code> . Row-wise values must sum up to 1 and must be in the interval [0,1].
grouping	Class vector.
dis	Logical, whether to optimize the dispersion parameter in <code>pbeta</code> .
eps	Minimum variation of membership values. If variance is smaller than <code>eps</code> , the values are treated as one point.

Details

With `betascale` and `b.scal`, membership values of an argmax classifier are scaled in such a way, that the mean membership value of those values which are assigned to each class reflect the mean correctness rate of that values. This is done via `qbeta` and `pbeta` with the appropriate shape parameters. If `dis` is `TRUE`, it is tried that the variation of membership values is optimal for the accuracy relative to the correctness rate. If the variation of the membership values is less than `eps`, they are treated as one point and shifted towards the correctness rate.

Value

A list containing

model	Estimated parameters for <code>betascale</code> .
eps	Value of <code>eps</code> from the call.
member	Scaled membership values.

Author(s)

Karsten Luebke, Uwe Ligges

References

Garczarek, Ursula Maria (2002): Classification rules in standardized partition spaces. Dissertation, University of Dortmund. URL <http://hdl.handle.net/2003/2789>

See Also

`betascale`, `e.scal`

Examples

```

library(MASS)
data(B3)
pB3 <- predict(lda(PHASEN ~ ., data = B3))$posterior
pbB3 <- b.scal(pB3, B3$PHASEN, dis = TRUE)
ucpm(pB3, B3$PHASEN)
ucpm(pbB3$member, B3$PHASEN)

```

B3

*West German Business Cycles 1955-1994***Description**

West German Business Cycles 1955-1994

Usage

```
data(B3)
```

Format

A data frame with 157 observations on the following 14 variables.

PHASEN a factor with levels 1 (upswing), 2 (upper turning points), 3 (downswing), and 4 (lower turning points).

BSP91JW GNP (y)

CP91JW Private Consumption (y)

DEFRATE Government deficit (percent of GNP)

EWAJW Wage and salary earners (y)

EXIMRATE Net exports as (percent of GNP)

GM1JW Money supply M1 (y)

IAU91JW Investment in equipment (y)

IB91JW Investment in construction (y)

LSTKJW Unit labor cost (y)

PBSPJW GNP price deflator (y)

PCPJW Consumer price index (y)

ZINSK Short term interest rate (nominal)

ZINSLR Long term interest rate (real)

where (y) stands for “yearly growth rates”.

Note that years and corresponding year quarters are given in the row names of the data frame, e.g. “1988,3” for the third quarter in 1988.

Details

The West German Business Cycles data (1955-1994) is analyzed by the project *B3* of the SFB475 (Collaborative Research Centre “Reduction of Complexity for Multivariate Data Structures”), supported by the Deutsche Forschungsgemeinschaft.

Source

RWI (Rheinisch Westfälisches Institut für Wirtschaftsforschung), Essen, Germany.

References

Heilemann, U. and Münch, H.J. (1996): West German Business Cycles 1963-1994: A Multivariate Discriminant Analysis. *CIRET-Conference in Singapore, CIRET-Studien 50*.

See Also

For benchmarking on this data see also [benchB3](#)

Examples

```
data(B3)
summary(B3)
```

benchB3

Benchmarking on B3 data

Description

Evaluates the performance of a classification method on the [B3](#) data.

Usage

```
benchB3(method, prior = rep(1/4, 4), sv = "4", scale = FALSE, ...)
```

Arguments

<code>method</code>	classification method to use
<code>prior</code>	prior probabilities of classes
<code>sv</code>	class of the start of a business cycle
<code>scale</code>	logical, whether to use <code>scale</code> first
<code>...</code>	further arguments passed to <code>method</code>

Details

The performance of classification methods on cyclic data can be measured by a special form of cross-validation: Leave-One-Cycle-Out. That means that a complete cycle is used as test data and the others are used as training data. This is repeated for all complete cycles in the data.

Value

A list with elements

MODEL	list with the model returned by method of the training data
error	vector of test error rates in cycles
llco.error	leave-one-cycle-out error rate

Author(s)

Karsten Luebke

See Also

[B3](#)

Examples

```
perLDA <- benchB3("lda")
## Not run:
## due to parameter optimization rda takes a while
perRDA <- benchB3("rda")
library(rpart)
## rpart will not work with prior argument:
perRpart <- benchB3("rpart", prior = NULL)
## End(Not run)
```

betascale

Scale membership values according to a beta scaling

Description

Performs the scaling for beta scaling learned by [b.scal](#).

Usage

```
betascale(betaobj, member)
```

Arguments

betaobj	A model learned by b.scal .
member	Membership values to be scaled.

Details

See [b.scal](#).

Value

A matrix with the scaled membership values.

See Also

[b.scal](#), [e.scal](#)

Examples

```
library(MASS)
data(B3)
pB3 <- predict(lda(PHASEN ~ ., data = B3))$posterior
pbB3 <- b.scal(pB3, B3$PHASEN)
betascale(pbB3)
```

calc.trans

Calculation of transition probabilities

Description

Function to estimate the probabilities of a time series to stay or change the state.

Usage

```
calc.trans(x)
```

Arguments

x (factor) vector of states

Details

To estimate the transition probabilities the empirical frequencies are counted.

Value

The transition probabilities matrix. $x[i, j]$ is the probability to change from state i to state j .

Author(s)

Karsten Luebke

Examples

```
data(B3)
calc.trans(B3$PHASEN)
```

centerlines

Lines from classborders to the center

Description

Function which constructs the lines from the borders between two classes to the center. To be used in connection with `triplot` and `quadplot`.

Usage

```
centerlines(n)
```

Arguments

`n` number of classes. Meaningful are 3 or 4.

Value

a matrix with `n`-columns.

Author(s)

Karsten Luebke

See Also

`triplot`, `quadplot`

Examples

```
centerlines(3)
centerlines(4)
```

classscatter*Classification scatterplot matrix*

Description

Function to plot a scatterplot matrix with a classification result.

Usage

```
classscatter(formula, data, method, col.correct = "black",
             col.wrong = "red", gs = NULL, ...)
```

Arguments

<code>formula</code>	formula of the form <code>groups ~ x1 + x2 + . . .</code> . That is, the response is the grouping factor and the right hand side specifies the (non-factor) discriminators.
<code>data</code>	Data frame from which variables specified in formula are preferentially to be taken.
<code>method</code>	character, name of classification function (e.g. "lda").
<code>col.correct</code>	color to use for correct classified objects.
<code>col.wrong</code>	color to use for missclassified objects.
<code>gs</code>	group symbol (plot character), must have the same length as the data. If NULL, <code>as.character(groups)</code> is the default.
<code>. . .</code>	further arguments passed to the underlying classification method or plot functions.

Value

The actual error rate.

Author(s)

Karsten Luebke

See Also

[plot](#)

Examples

```
data(B3)
library(MASS)
classscatter(PHASEN ~ BSP91JW + EWAJW + LSTKJW,
             data = B3, method = "lda")
```

`corclust`

Function to identify groups of highly correlated variables for removing correlated features from the data for further analysis.

Description

A hierarchical clustering of variables using `hclust` is performed using 1 - the absolute correlation as a distance measure between two variables.

Usage

```
corclust(x, cl = NULL, mincor = NULL, prnt = FALSE, method = "complete")
```

Arguments

<code>x</code>	Either a data frame or a matrix consisting of numerical attributes.
<code>cl</code>	Optional vector of ty factor indicating class levels, if class specific correlations should to be considered.
<code>mincor</code>	Optional vector of degrees of correlation within a cluster of variables that will be indicated in the plot by a line.
<code>prnt</code>	Logical indicating whether the matrix of distances should be printed.
<code>method</code>	Linkage to be used for clustering. Default is <code>complete</code> linkage.

Details

The main output consists in the tree visualization of the clustered variables. Each cluster consists of a set of correlated variables according to the chosen clustering criterion. The default criterion is 'complete'. This choice is meaningful as it represents the *minimum absolute correlation* between all variables of a cluster. Further proceeding would consist in choosing one variable of each cluster to obtain a subset of rather uncorrelated variables for further analysis. If an additional class vector `cl` is given to the function for any two variables their minimum correlation over all classes is used.

Value

<code>min.abs.cor</code>	Matrix of distances used for clustering containing 1 - the absolute correlation between any two variables.
<code>clustering</code>	Result object of the hierarchical clustering.

Author(s)

Gero Szepannek

See Also

See also [hclust](#), for details on the clustering algorithm.

Examples

```
data(iris)
classes <- iris$Species
variables <- iris[,1:4]
corclust(variables, classes, mincor = 0.6)
```

`countries`*Socioeconomic data for the most populous countries.*

Description

Socioeconomic data for the most populous countries.

Usage

```
data(countries)
```

Format

A data frame with 42 observations on the following 7 variables.

Country name of the country.

Popul population.

PopDens population density.

GDPpp GDP per inhabitant.

LifeEx mean life expectation

InfMor infant mortality

Illit illiteracy rate

Source

CIA World Factbook <http://www.cia.gov/cia/publications/factbook/>

Examples

```
data(countries)
summary(countries)
```

`dkernel`*Estimate density of a given kernel*

Description

Given an estimated kernel density this function estimates the density of a new vector.

Usage

```
dkernel(x, kernel = density(x), interpolate = FALSE, ...)
```

Arguments

x	vector of which the density should be estimated
kernel	object of class <code>density</code>
interpolate	Interpolate or use <code>density</code> of nearest point?
...	currently not used.

Value

Density of x in kernel.

Author(s)

Karsten Luebke

See Also

`density`, `NaiveBayes`

Examples

```
kern <- density(rnorm(50))
x <- seq(-3, 3, len = 100)
y <- dkernel(x, kern)
plot(x, y, type = "l")
```

drawpart

Plotting the 2-d partitions of classification methods

Description

Plot showing the classification of observations based on classification methods (e.g. `lda`, `qda`) for two variables. Moreover, the classification borders are displayed and the apparent error rates are given in each title.

Usage

```
drawpart(grouping, x, y, method = "lda", prec = 100, xlab = NULL,
  ylab = NULL, col.correct = "black", col.wrong = "red",
  col.mean = "black", col.contour = "darkgrey",
  gs = as.character(grouping), pch.mean = 19, cex.mean = 1.3,
  print.err = 0.7, legend.err = FALSE, legend.bg = "white",
  imageplot = TRUE, image.colors = cm.colors(nc),
  plot.control = list(), ...)
```

Arguments

<code>grouping</code>	factor specifying the class for each observation.
<code>x</code>	first explanatory vector.
<code>y</code>	second explanatory vector.
<code>method</code>	the method the classification is based on, currently supported are: <code>lda</code> , <code>qda</code> , <code>rpart</code> , <code>naiveBayes</code> , <code>rda</code> , <code>sknn</code> and <code>svmlight</code>
<code>prec</code>	precision used to draw the classification borders (the higher the more precise; default: 100).
<code>xlab</code>	a title for the x axis.
<code>ylab</code>	a title for the y axis.
<code>col.correct</code>	color for correct classified objects.
<code>col.wrong</code>	color for wrong classified objects.
<code>col.mean</code>	color for class means (only for methods <code>lda</code> and <code>qda</code>).
<code>col.contour</code>	color of the contour lines (if <code>imageplot = FALSE</code>).
<code>gs</code>	group symbol (plot character), must have the same length as <code>grouping</code> .
<code>pch.mean</code>	plot character for class means (only for methods <code>lda</code> and <code>qda</code>).
<code>cex.mean</code>	character expansion for class means (only for methods <code>lda</code> and <code>qda</code>).
<code>print.err</code>	character expansion for text specifying the apparent error rate. If <code>print.err = 0</code> , nothing is printed.
<code>legend.err</code>	logical; whether to plot the apparent error rate above the plot (if <code>FALSE</code>), or into a legend into the upper right corner of the plot (if <code>TRUE</code>). This argument is ignored, if <code>print.err = 0</code> , i.e. if no error rate is printed.
<code>legend.bg</code>	Background colour to use for the legend.
<code>imageplot</code>	logical; whether to use an <code>image</code> plot or <code>contour</code> lines.
<code>image.colors</code>	colors used for the <code>imageplot</code> , if <code>TRUE</code> .
<code>plot.control</code>	A list containing further arguments passed to the underlying plot functions.
<code>...</code>	Further arguments passed to the classification method.

Author(s)

Karsten Luebke, (luebke@statistik.tu-dortmund.de), Uwe Ligges, Irina Czogiel

See Also

[partimat](#)

`e.scal`*Function to calculate e- or softmax scaled membership values*

Description

Calculates the e- or softmax scaled membership values of an argmax based classification rule.

Usage

```
e.scal(x, k = 1, tc = NULL)
```

Arguments

<code>x</code>	matrix of membership values
<code>k</code>	parameter for e-scaling (1 for softmax)
<code>tc</code>	vector of true classes (required if <code>k</code> has to be optimized)

Details

For any membership vector y $\exp(y \cdot k) / \sum \exp(y \cdot k)$ is calculated. If $k=1$, the classical softmax scaling is used. If the true classes are given, k is optimized so that the apparent error rate is minimized.

Value

A list containing elements

<code>sv</code>	Scaled values
<code>k</code>	Optimal k

Author(s)

Karsten Luebke, (luebke@statistik.tu-dortmund.de)

References

Garczarek, Ursula Maria (2002): Classification rules in standardized partition spaces. Dissertation, University of Dortmund. URL <http://hdl.handle.net/2003/2789>

Examples

```
library(MASS)
data(iris)
ldaobj <- lda(Species ~ ., data = iris)
ldapred <- predict(ldaobj)$posterior
e.scal(ldapred)
e.scal(ldapred, tc = iris$Species)
```

Description

Produces an object of class EDAM which is a two dimensional representation of data in a rectangular, equally spaced grid as known from Self-Organizing Maps.

Usage

```
EDAM(EVO, nzx = 0, iter.max = 10, random = TRUE, standardize = FALSE,
     wgths = 0, classes = 0, sa = TRUE, temp.in = 0.5, temp.fin = 1e-07,
     temp.gamma = 0)
```

Arguments

EVO	either a symmetric dissimilarity matrix or a matrix of arbitrary dimensions whose n rows correspond to cases and whose k columns correspond to variables.
nxz	an integer specifying the number of vertical bars in the grid. By default, nxz is chosen automatically, so that the grid gets closest do a square. If n is no multiple of nxz, all surplus objects are skipped.
iter.max	an integer giving the maximum number of iterations to perform for the same neighborhood size.
random	logical. If TRUE, the initial order is drawn from a uniform distribution.
standardize	logical. If TRUE, the measurements in EVO are standardized before calculating Euclidean distances. Measurements are standardized for each variable by dividing by the variable's standard deviation. Meaningless if EVO is a dissimilarity matrix.
wgths	an optional vector of length k giving relative weights of the variables in computing Euclidean distances. Meaningless if EVO is a dissimilarity matrix.
classes	an optional vector of length n specifying the membership to classes for all objects.
sa	logical. If TRUE, the optimization is obtained by Simulated Annealing.
temp.in	numeric giving the initial temperature, if sa is set to TRUE.
temp.fin	numeric giving the final temperature, if sa is set to TRUE. Meaningless if temp.gamma is greater than 0.
temp.gamma	numeric giving the relative change of the temperature from one iteration to the other, if sa is set to TRUE.

Details

The data given by `EVO` is visualized by the EDAM-algorithm. This method approximates the best visualization where goodness is measured by S , a transformation of the criterion `stress` as i.e. known from `sammon`. The target space of the visualization is restricted to a grid so the problem has a discrete solution space. Originally this restriction was made to make the results comparable to those of Kohonen Self-Organizing Maps. But it turns out that also for reasons of a clear arrangement the representation in a grid can be more favorable than in the hole plane.

During the computation of EDAM 3 values indicating its progress are given online. The first is the number of the actual iteration, the second the maximum number of overall performed iterations. The latter may reduce during computation, since the neighborhood reduces in case of convergence before the last iteration. The last number gives the actual criterion S . The default plot method `plot.edam` for objects of class `EDAM` is `shardsplot`.

Value

EDAM returns an object of class `EDAM`, which is a list containing the following components:

<code>preimages</code>	the re-ordered data; the position of the i -th object is where Z equals i .
<code>Z</code>	a matrix representing the positions of the <code>preimages</code> in the grid by their numbers.
<code>Z.old.terms</code>	a matrix representing the positions of the data in original order in the grid by their numbers.
<code>cl.ord</code>	a vector giving the re-ordered classes. All elements equal 1 if argument <code>classes</code> is undefined.
<code>S</code>	the criterion of the map

Author(s)

Nils Raabe

References

Raabe, N. (2003). *Vergleich von Kohonen Self-Organizing-Maps mit einem nichtsimultanen Klassifikations- und Visualisierungsverfahren*. Diploma Thesis, Department of Statistics, University of Dortmund.
<http://www.statistik.tu-dortmund.de/de/content/einrichtungen/lehrstuehle/personen/raabe/Diplomarbeit.pdf>.

See Also

`shardsplot`, `TopoS`

Examples

```
# Compute an Eight Directions Arranged Map for a random sample
# of the iris data.
data(iris)
set.seed(1234)
iris.sample <- sample(150, 42)
```

```

irisEDAM <- EDAM(iris[iris.sample, 1:4], classes = iris[iris.sample, 5],
  standardize = TRUE, iter.max = 3)
plot(irisEDAM, vertices = FALSE)
legend(3, 5, col = rainbow(3), legend = levels(iris[,5]), pch = 16)
print(irisEDAM)

# Construct clusters within the phases of the german business data
# and visualize the centroids by EDAM.
data(B3)
phasemat <- lapply(1:4, function(x) B3[B3[,1] == x, 2:14])
subclasses <- lapply(phasemat,
  function(x) cutree(hclust(dist(x)), k = round(nrow(x) / 4.47)))
centroids <- lapply(1:4,
  function(y) apply(phasemat[[y]], 2,
    function(x) by(x, subclasses[[y]], mean)))
centmat <- matrix(unlist(sapply(centroids, t)), ncol = 13,
  byrow = TRUE, dimnames = list(NULL, colnames(centroids[[1]]))
centclasses <- unlist(lapply(1:4,
  function(x) rep(x, unlist(lapply(centroids, nrow))[x])))
B3EDAM <- EDAM(centmat, classes = centclasses, standardize = TRUE,
  iter.max = 6, rand = FALSE)
plot(B3EDAM, standardize = TRUE)
opar <- par(xpd = NA)
legend(4, 5.1, col = rainbow(4), pch = 16, xjust = 0.5, yjust = 0,
  ncol = 2, legend = c("upswing", "upper turning point",
    "downswing", "lower turning point"))

print(B3EDAM)
par(opar)

```

errormatrix

Tabulation of prediction errors by classes

Description

Cross-tabulates true and predicted classes with the option to show relative frequencies.

Usage

```
errormatrix(true, predicted, relative = FALSE)
```

Arguments

true	Vector of true classes.
predicted	Vector of predicted classes.
relative	Logical. If TRUE rows are normalized to show relative frequencies (see below).

Details

Given vectors of true and predicted classes, a (symmetric) table of misclassifications is constructed. Element [i,j] shows the number of objects of class i that were classified as class j; so the main diagonal shows the correct classifications. The last row and column show the corresponding sums of misclassifications, the lower right element is the total sum of misclassifications.

If 'relative' is TRUE, the *rows* are normalized so they show relative frequencies instead. The lower right element now shows the total error rate, and the remaining last row sums up to one, so it shows "where the misclassifications went".

Value

A (named) matrix.

Note

Concerning the case that 'relative' is TRUE:

If a prior distribution over the classes is given, the misclassification rate that is returned as the lower right element (which is only the fraction of misclassified *data*) is not an estimator for the expected misclassification rate.

In that case you have to multiply the individual error rates for each class (returned in the last column) with the corresponding prior probabilities and sum these up (see example below).

Both error rate estimates are equal, if the fractions of classes in the data are equal to the prior probabilities.

Author(s)

Christian Röver, (roever@statistik.tu-dortmund.de)

See Also

[table](#)

Examples

```
data(iris)
library(MASS)
x <- lda(Species ~ Sepal.Length + Sepal.Width, data=iris)
y <- predict(x, iris)

# absolute numbers:
errormatrix(iris$Species, y$class)

# relative frequencies:
errormatrix(iris$Species, y$class, relative = TRUE)

# percentages:
round(100 * errormatrix(iris$Species, y$class, relative = TRUE), 0)

# expected error rate in case of class prior:
```

```

indiv.rates <- errormatrix(iris$Species, y$class, relative = TRUE)[1:3, 4]
prior <- c("setosa" = 0.2, "versicolor" = 0.3, "virginica" = 0.5)
total.rate <- t(indiv.rates) %*% prior
total.rate

```

friedman.data

Friedman's classification benchmark data

Description

Function to generate 3-class classification benchmarking data as introduced by J.H. Friedman (1989)

Usage

```

friedman.data(setting = 1, p = 6, samplesize = 40, asmatrix = FALSE)

```

Arguments

<code>setting</code>	the problem setting (integer 1,2,...,6).
<code>p</code>	number of variables (6, 10, 20 or 40).
<code>samplesize</code>	sample size (number of observations, >=6).
<code>asmatrix</code>	if TRUE, results are returned as a matrix, otherwise as a data frame (default).

Details

When J.H. Friedman introduced the Regularized Discriminant Analysis ([rda](#)) in 1989, he used artificially generated data to test the procedure and to examine its performance in comparison to Linear and Quadratic Discriminant Analysis (see also [lda](#) and [qda](#)).

6 different settings were considered to demonstrate potential strengths and weaknesses of the new method:

1. equal spherical covariance matrices,
2. unequal spherical covariance matrices,
3. equal, highly ellipsoidal covariance matrices with mean differences in low-variance subspace,
4. equal, highly ellipsoidal covariance matrices with mean differences in high-variance subspace,
5. unequal, highly ellipsoidal covariance matrices with zero mean differences and
6. unequal, highly ellipsoidal covariance matrices with nonzero mean differences.

For each of the 6 settings data was generated with 6, 10, 20 and 40 variables.

Classification performance was then measured by repeatedly creating training-datasets of 40 observations and estimating the misclassification rates by test sets of 100 observations.

The number of classes is always 3, class labels are assigned randomly (with equal probabilities) to observations, so the contributions of classes to the data differs from dataset to dataset. To make sure covariances can be estimated at all, there are always at least two observations from each class in a dataset.

Value

Depending on `asmatrix` either a data frame or a matrix with `samplesize` rows and `p+1` columns, the first column containing the class labels, the remaining columns being the variables.

Author(s)

Christian Röver, roever@statistik.tu-dortmund.de

References

Friedman, J.H. (1989): Regularized Discriminant Analysis. In: *Journal of the American Statistical Association* 84, 165-175.

See Also

[rda](#)

Examples

```
# Reproduce the 1st setting with 6 variables.
# Error rate should be somewhat near 9 percent.
training <- friedman.data(1, 6, 40)
x <- rda(class ~ ., data = training, gamma = 0.74, lambda = 0.77)
test <- friedman.data(1, 6, 100)
y <- predict(x, test[, -1])
errormatrix(test[, 1], y$class)
```

greedy.wilks

Stepwise forward variable selection for classification

Description

Performs a stepwise forward variable/model selection using the Wilk's Lambda criterion.

Usage

```
greedy.wilks(X, ...)  
## Default S3 method:  
greedy.wilks(X, grouping, niveau = 0.2, ...)  
## S3 method for class 'formula':  
greedy.wilks(formula, data = NULL, ...)
```

Arguments

<code>X</code>	matrix or data frame (rows=cases, columns=variables)
<code>grouping</code>	class indicator vector
<code>formula</code>	formula of the form 'groups ~ x1 + x2 + ...'
<code>data</code>	data frame (or matrix) containing the explanatory variables
<code>niveau</code>	level for the approximate F-test decision
<code>...</code>	further arguments to be passed to the default method, e.g. <code>niveau</code>

Details

A stepwise forward variable selection is performed. The initial model is defined by starting with the variable which separates the groups most. The model is then extended by including further variables depending on the Wilk's lambda criterion: Select the one which minimizes the Wilk's lambda of the model including the variable if its p-value still shows statistical significance.

Value

A list of two components, a formula of the form 'response ~ list + of + selected + variables', and a `data.frame` `results` containing the following variables:

<code>vars</code>	the names of the variables in the final model in the order of selection.
<code>Wilks.lambda</code>	the appropriate Wilks' lambda for the selected variables.
<code>F.statistics.overall</code>	the approximated F-statistic for the so far selected model.
<code>p.value.overall</code>	the appropriate p-value of the F-statistic.
<code>F.statistics.diff</code>	the approximated F-statistic of the partial Wilks's lambda (for comparing the model including the new variable with the model not including it).
<code>p.value.diff</code>	the appropriate p-value of the F-statistic of the partial Wilk's lambda.

Author(s)

Andrea Preusser, Karsten Luebke

References

Mardia, K. V. , Kent, J. T. and Bibby, J. M. (1979), *Multivariate analysis*, Academic Press (New York; London)

See Also

[stepclass](#), [manova](#)

Examples

```
data(B3)
gw_obj <- greedy.wilks(PHASEN ~ ., data = B3, niveau = 0.1)
gw_obj
## now you can say stuff like
## lda(gw_obj$formula, data = B3)
```

hmm.sop

Calculation of HMM Sum of Path

Description

A Hidden Markov Model for the classification of states in a time series. Based on the transition probabilities and the so called emission probabilities ($p(class|x)$) the ‘prior probabilities’ of states (classes) in time period t given all past information in time period t are calculated.

Usage

```
hmm.sop(sv, trans.matrix, prob.matrix)
```

Arguments

sv state at time 0
trans.matrix matrix of transition probabilities
prob.matrix matrix of $p(class|x)$

Value

Returns the ‘prior probabilities’ of states.

Author(s)

Daniel Fischer, Reinald Oetsch

References

Garczarek, Ursula Maria (2002): Classification rules in standardized partition spaces. Dissertation, University of Dortmund. URL <http://hdl.handle.net/2003/2789>

See Also

[calc.trans](#)

Examples

```

library(MASS)
data(B3)
trans.matrix <- calc.trans(B3$PHASEN)

# Calculate posterior prob. for the classes via lda
prob.matrix <- predict(lda(PHASEN ~ ., data = B3))$post
errormatrix(B3$PHASEN, apply(prob.matrix, 1, which.max))
prior.prob <- hmm.sop("2", trans.matrix, prob.matrix)
errormatrix(B3$PHASEN, apply(prior.prob, 1, which.max))

```

loclda

*Localized Linear Discriminant Analysis (LocLDA)***Description**

A localized version of Linear Discriminant Analysis.

Usage

```

loclda(x, ...)

## S3 method for class 'formula':
loclda(formula, data, ..., subset, na.action)

## Default S3 method:
loclda(x, grouping, weight.func = function(x) 1/exp(x),
       k = nrow(x), weighted.apriori = TRUE, ...)

## S3 method for class 'data.frame':
loclda(x, ...)

## S3 method for class 'matrix':
loclda(x, grouping, ..., subset, na.action)

```

Arguments

formula	Formula of the form ‘groups ~ x1 + x2 + ...’.
data	Data frame from which variables specified in formula are to be taken.
x	Matrix or data frame containing the explanatory variables (required, if formula is not given).
grouping	(required if no formula principal argument is given.) A factor specifying the class for each observation.
weight.func	Function used to compute local weights. Must be finite over the interval [0,1]. See Details below.

<code>k</code>	Number of nearest neighbours used to construct localized classification rules. See Details below.
<code>weighted.apriori</code>	Logical: if TRUE, class prior probabilities are computed using local weights (see Details below). If FALSE, equal priors for all classes actually occurring in the train data are used.
<code>subset</code>	An index vector specifying the cases to be used in the training sample.
<code>na.action</code>	A function to specify the action to be taken if NAs are found. The default action is for the procedure to fail. An alternative is <code>na.omit</code> which leads to rejection of cases with missing values on any required variable.
<code>...</code>	Further arguments to be passed to <code>loclda.default</code> .

Details

This is an approach to apply the concept of localization described by Tutz and Binder (2005) to Linear Discriminant Analysis. The function `loclda` generates an object of class `loclda` (see Value below). As localization makes it necessary to build an individual decision rule for each test observation, this rule construction has to be handled by `predict.loclda`. For convenience, the rule building procedure is still described here.

To classify a test observation x_s , only the k nearest neighbours of x_s within the train data are used. Each of these k train observations $x_i, i = 1, \dots, k$, is assigned a weight w_i according to

$$w_i = K \left(\frac{\|x_i - x_s\|}{d_k} \right), i = 1, \dots, k$$

where K is the weighting function given by `weight.func`, $\|x_i - x_s\|$ is the euclidian distance of x_i and x_s and d_k is the euclidian distance of x_s to its k -th nearest neighbour. With these weights for each class $A_g, g = 1, \dots, G$, its weighted empirical mean $\hat{\mu}_g$ and weighted empirical covariance matrix are computed. The estimated pooled (weighted) covariance matrix $\hat{\Sigma}$ is then calculated from the individual weighted empirical class covariance matrices. If `weighted.apriori` is TRUE (the default), prior class probabilities are estimated according to:

$$prior_g := \frac{\sum_{i=1}^k (w_i \cdot I(x_i \in A_g))}{\sum_{i=1}^k (w_i)}$$

where I is the indicator function. If FALSE, equal priors for all classes are used. In analogy to Linear Discriminant Analysis, the decision rule for x_s is

$$\hat{A} := \operatorname{argmax}_{g \in 1, \dots, G} (posterior_g)$$

where

$$posterior_g := prior_g \cdot \exp \left(\left(-\frac{1}{2} \right) t(x_s - \hat{\mu}_g) \hat{\Sigma}^{-1} (x_s - \hat{\mu}_g) \right)$$

If $posterior_g < 10^{(-150)} \forall g \in \{1, \dots, G\}$, $posterior_g$ is set to $\frac{1}{G}$ for all $g \in 1, \dots, G$ and the test observation x_s is simply assigned to the class whose weighted mean has the lowest euclidian distance to x_s .

Value

A list of class `loclda` containing the following components:

<code>call</code>	The (matched) function call.
<code>learn</code>	Matrix containing the values of the explanatory variables for all train observations.
<code>grouping</code>	Factor specifying the class for each train observation.
<code>weight.func</code>	Value of the argument <code>weight.func</code> .
<code>k</code>	Value of the argument <code>k</code> .
<code>weighted.apriori</code>	Value of the argument <code>weighted.apriori</code> .

Author(s)

Marc Zentgraf (marc-zentgraf@gmx.de) and Karsten Luebke (luebke@statistik.tu-dortmund.de)

References

Tutz, G. and Binder, H. (2005): Localized classification. *Statistics and Computing* 15, 155-166.

See Also

[predict.loclda](#), [lda](#)

Examples

```
benchB3("lda")$l1co.error
benchB3("loclda")$l1co.error
```

locpvs

Pairwise variable selection for classification in local models

Description

Performs pairwise variable selection on subclasses.

Usage

```
locpvs(x, subclasses, subclass.labels, prior=NULL, method="lda",
       vs.method = c("ks.test", "stepclass", "greedy.wilks"),
       niveau=0.05, fold=10, impr=0.1, direct="backward", out=FALSE, ...)
```

Arguments

<code>x</code>	matrix or data frame containing the explanatory variables. <code>x</code> must consist of numerical data only.
<code>subclasses</code>	vector indicating the subclasses (a factor)
<code>subclass.labels</code>	must be a matrix with 2 columns, where the first column specifies the subclass and the second column the according upper class
<code>prior</code>	prior probabilities for the classes. If not specified the prior probabilities will be set according to proportion in “subclasses”. If specified the order of prior probabilities must be the same as in “subclasses”.
<code>method</code>	character, name of classification function (e.g. “lda” (default)).
<code>vs.method</code>	character, name of variable selection method. Must be one of “ks.test” (default), “stepclass” or “greedy.wilks”.
<code>niveau</code>	used niveau for “ks.test”
<code>fold</code>	parameter for cross-validation, if “stepclass” is chosen ‘vs.method’
<code>impr</code>	least improvement of performance measure desired to include or exclude any variable (≤ 1), if “stepclass” is chosen ‘vs.method’
<code>direct</code>	direction of variable selection, if “stepclass” is chosen ‘vs.method’. Must be one of “forward”, “backward” (default) or “both”.
<code>out</code>	indicator (logical) for textoutput during computation (slows down computation!), if “stepclass” is chosen ‘vs.method’
<code>...</code>	further parameters passed to classification function (‘method’) or variable selection method (‘vs.method’)

Details

A call on `pvs` is performed using “subclasses” as grouping variable. See `pvs` for further details.

Value

An object of class ‘locpvs’ containing the following components:

<code>pvs.result</code>	the complete output of the call to <code>pvs</code> (see <code>pvs</code> for further details)
<code>subclass.labels</code>	the subclass.labels as specified in function call

Author(s)

Gero Szepannek, (szepannek@statistik.tu-dortmund.de), Christian Neumann

References

Szepannek, G. and Weihs, C. (2006) Local Modelling in Classification on Different Feature Subspaces. In *Advances in Data Mining.*, ed Perner, P., LNAI 4065, pp. 226-234. Springer, Heidelberg.

See Also

[predict.locpvs](#) for predicting 'locpvs' models and [pvs](#)

Examples

```
## this example might be a bit artificial, but it sufficiently shows how locpvs has to be us

## learn a locpvs-model on the Vehicle dataset

library("mlbench")
data("Vehicle")

subclass <- Vehicle$Class # use four car-types in dataset as subclasses
## aggregate "bus" and "van" to upper-class "big" and "saab" and "opel" to upper-class "small"
subclass_class <- matrix(c("bus", "van", "saab", "opel", "big", "big", "small", "small"), ncol=2)

## learn now a locpvs-model for the subclasses:
model <- locpvs(Vehicle[,1:18], subclass, subclass_class)
model # short summary, showing the class-pairs of the submodels
# together with the selected variables and the relation of sub- to upperclasses

## predict:
pred <- predict(model, Vehicle[,1:18])

## now you can look at the predicted classes:
pred$class
## or at the posterior probabilities:
pred$posterior
## or at the posterior probabilities for the subclasses:
pred$subclass.posterior
```

meclight.default *Minimal Error Classification*

Description

Computer intensive method for linear dimension reduction that minimizes the classification error directly.

Usage

```
meclight(x, ...)

## Default S3 method:
meclight(x, grouping, r = 1, fold = 10, ...)
## S3 method for class 'formula':
meclight(formula, data = NULL, ..., subset, na.action = na.fail)
```

```
## S3 method for class 'data.frame':
meclight(x, ...)
## S3 method for class 'matrix':
meclight(x, grouping, ..., subset, na.action = na.fail)
```

Arguments

<code>x</code>	(required if no formula is given as the principal argument.) A matrix or data frame containing the explanatory variables.
<code>grouping</code>	(required if no formula principal argument is given.) A factor specifying the class for each observation.
<code>r</code>	Dimension of projected subspace.
<code>fold</code>	Number of Bootstrap samples.
<code>formula</code>	A formula of the form <code>groups ~ x1 + x2 + ...</code> . That is, the response is the grouping factor and the right hand side specifies the (non-factor) discriminators.
<code>data</code>	Data frame from which variables specified in formula are preferentially to be taken.
<code>subset</code>	An index vector specifying the cases to be used in the training sample. (NOTE: If given, this argument must be named.)
<code>na.action</code>	A function to specify the action to be taken if NAs are found. The default action is for the procedure to fail. An alternative is <code>na.omit</code> , which leads to rejection of cases with missing values on any required variable. (NOTE: If given, this argument must be named.)
<code>...</code>	Further arguments passed to <code>lda</code> .

Details

Computer intensive method for linear dimension reduction that minimizes the classification error in the projected subspace directly. Classification is done by `lda`. In contrast to the reference function minimization is done by Nelder-Mead in `optim`.

Value

<code>method.model</code>	An object of class 'lda'.
<code>Proj.matrix</code>	Projection matrix.
<code>B.error</code>	Estimated bootstrap error rate.
<code>B.impro</code>	Improvement in <code>lda</code> error rate.

Author(s)

Maria Eveslage, Karsten Luebke

References

Roehl, M.C., Weihs, C., and Theis, W. (2002): Direct Minimization in Multivariate Classification. *Computational Statistics*, 17, 29-46.

See Also

[predict.meclight](#)

Examples

```
data(iris)
meclight.obj <- meclight(Species ~ ., data = iris)
meclight.obj
```

NaiveBayes

Naive Bayes Classifier

Description

Computes the conditional a-posterior probabilities of a categorical class variable given independent predictor variables using the Bayes rule.

Usage

```
## S3 method for class 'formula':
NaiveBayes(formula, data, ..., subset, na.action = na.pass)
## Default S3 method:
NaiveBayes(x, grouping, prior, usekernel = FALSE, fL = 0, ...)
```

Arguments

<code>x</code>	a numeric matrix, or a data frame of categorical and/or numeric variables.
<code>grouping</code>	class vector (a factor).
<code>formula</code>	a formula of the form <code>class ~ x1 + x2 + ...</code> . Interactions are not allowed.
<code>data</code>	a data frame of predictors (caegorical and/or numeric).
<code>prior</code>	the prior probabilities of class membership. If unspecified, the class proportions for the training set are used. If present, the probabilities should be specified in the order of the factor levels.
<code>usekernel</code>	if TRUE a kernel density estimate (density) is used for density estimation. If FALSE a normal density is estimated.
<code>fL</code>	Factor for Laplace correction, default factor is 0, i.e. no correction.
<code>...</code>	arguments passed to density .
<code>subset</code>	for data given in a data frame, an index vector specifying the cases to be used in the training sample. (NOTE: If given, this argument must be named.)
<code>na.action</code>	a function to specify the action to be taken if NAs are found. The default action is not to count them for the computation of the probability factors. An alternative is <code>na.omit</code> , which leads to rejection of cases with missing values on any required variable. (NOTE: If given, this argument must be named.)

Details

This implementation of Naive Bayes as well as this help is based on the code by David Meyer in the package `e1071` but extended for kernel estimated densities and user specified `prior` probabilities. The standard naive Bayes classifier (at least this implementation) assumes independence of the predictor variables.

Value

An object of class "NaiveBayes" including components:

<code>apriori</code>	Class distribution for the dependent variable.
<code>tables</code>	A list of tables, one for each predictor variable. For each categorical variable a table giving, for each attribute level, the conditional probabilities given the target class. For each numeric variable, a table giving, for each target class, mean and standard deviation of the (sub-)variable or a object of class <code>density</code> .

Author(s)

Karsten Luebke

See Also

`predict.NaiveBayes`, `plot.NaiveBayes`, `naiveBayes`, `qda`

Examples

```
data(iris)
m <- NaiveBayes(Species ~ ., data = iris)
```

nm

Nearest Mean Classification

Description

Function for nearest mean classification.

Usage

```
nm(x, ...)
```

```
## Default S3 method:
nm(x, grouping, gamma = 0, ...)
```

```
## S3 method for class 'data.frame':
nm(x, ...)
```

```
## S3 method for class 'matrix':
nm(x, grouping, ..., subset, na.action = na.fail)
```

```
## S3 method for class 'formula':
nm(formula, data = NULL, ..., subset, na.action = na.fail)
```

Arguments

<code>x</code>	matrix or data frame containing the explanatory variables (required, if <code>formula</code> is not given).
<code>grouping</code>	factor specifying the class for each observation (required, if <code>formula</code> is not given).
<code>formula</code>	formula of the form <code>groups ~ x1 + x2 + ...</code> . That is, the response is the grouping factor and the right hand side specifies the (non-factor) discriminators.
<code>data</code>	Data frame from which variables specified in <code>formula</code> are preferentially to be taken.
<code>gamma</code>	gamma parameter for rbf weight of the distance to mean. If <code>gamma=0</code> the posterior is 1 for the nearest class (mean) and 0 else.
<code>subset</code>	An index vector specifying the cases to be used in the training sample. (Note: If given, this argument must be named.)
<code>na.action</code>	specify the action to be taken if NAs are found. The default action is for the procedure to fail. An alternative is <code>na.omit</code> , which leads to rejection of cases with missing values on any required variable. (Note: If given, this argument must be named.)
<code>...</code>	

Details

`nm` is calling `sknn` with the class means as observations. If `gamma>0` a gaussian like density is used to weight the distance to the class means `weight=exp(-gamma*distance)`. This is similar to an rbf kernel. If the distances are large it may be useful to `scale` the data first.

Value

A list containing the function call and the class means (`learn`).

Author(s)

Karsten Luebke, {luebke@statistik.tu-dortmund.de}

See Also

[sknn](#), [rda](#), [knn](#)

Examples

```
data(B3)
x <- nm(PHASEN ~ ., data = B3)
x$learn
x <- nm(PHASEN ~ ., data = B3, gamma = 0.1)
predict(x)$post
```

 partimat

Plotting the 2-d partitions of classification methods

Description

Provides a multiple figure array which shows the classification of observations based on classification methods (e.g. `lda`, `qda`) for every combination of two variables. Moreover, the classification borders are displayed and the apparent error rates are given in each title.

Usage

```
partimat(x, ...)

## Default S3 method:
partimat(x, grouping, method = "lda", prec = 100,
         nplots.vert, nplots.hor, main = "Partition Plot", name, mar,
         plot.matrix = FALSE, plot.control = list(), ...)
## S3 method for class 'data.frame':
partimat(x, ...)
## S3 method for class 'matrix':
partimat(x, grouping, ..., subset, na.action = na.fail)
## S3 method for class 'formula':
partimat(formula, data = NULL, ..., subset, na.action = na.fail)
```

Arguments

<code>x</code>	matrix or data frame containing the explanatory variables (required, if <code>formula</code> is not given).
<code>grouping</code>	factor specifying the class for each observation (required, if <code>formula</code> is not given).
<code>formula</code>	formula of the form <code>groups ~ x1 + x2 + ...</code> . That is, the response is the grouping factor and the right hand side specifies the (non-factor) discriminators.
<code>method</code>	the method the classification is based on, currently supported are: <code>lda</code> , <code>qda</code> , <code>rpart</code> , <code>naiveBayes</code> , <code>rda</code> , <code>sknn</code> and <code>svmlight</code>
<code>prec</code>	precision used to draw the classification borders (the higher the more precise; default: 100).
<code>data</code>	Data frame from which variables specified in <code>formula</code> are preferentially to be taken.
<code>nplots.vert</code>	number of rows in the multiple figure array
<code>nplots.hor</code>	number of columns in the multiple figure array
<code>subset</code>	index vector specifying the cases to be used in the training sample. (Note: If given, this argument must be named.)

<code>na.action</code>	specify the action to be taken if NAs are found. The default action is for the procedure to fail. An alternative is <code>na.omit</code> , which leads to rejection of cases with missing values on any required variable. (Note: If given, this argument must be named.)
<code>main</code>	title
<code>name</code>	Variable names to be printed at the axis / into the diagonal.
<code>mar</code>	numerical vector of the form <code>c(bottom, left, top, right)</code> which gives the lines of margin to be specified on the four sides of the plot. Defaults are <code>rep(0, 4)</code> if <code>plot.matrix = TRUE</code> , <code>c(5, 4, 2, 1) + 0.1</code> otherwise.
<code>plot.matrix</code>	logical; if <code>TRUE</code> , like a scatterplot matrix; if <code>FALSE</code> (default) uses less space and arranges the plots “optimal” (using a fuzzy algorithm) in an array by plotting each pair of variables once.
<code>plot.control</code>	A list containing further arguments passed to the underlying plot functions (and to <code>drawpart</code>).
<code>...</code>	Further arguments passed to the classification method (through <code>drawpart</code>).

Note

Warnings such as ‘parameter “xyz” couldn’t be set in high-level plot function’ are expected, if making use of

Author(s)

Karsten Luebke, (luebke@statistik.tu-dortmund.de), Uwe Ligges, Irina Czogiel

See Also

for much more fine tuning see [drawpart](#)

Examples

```
library(MASS)
data(iris)
partimat(Species ~ ., data = iris, method = "lda")
## Not run:
partimat(Species ~ ., data = iris, method = "lda",
         plot.matrix = TRUE, imageplot = FALSE) # takes some time ...
## End(Not run)
```

`plineplot`*Plotting marginal posterior class probabilities*

Description

For a given variable the posteriori probabilities of the classes given by a classification method are plotted. The variable need not be used for the actual classification.

Usage

```
plineplot(formula, data, method, x, col.wrong = "red",
          ylim = c(0, 1), loo = FALSE, mfrow, ...)
```

Arguments

<code>formula</code>	formula of the form <code>groups ~ x1 + x2 + ...</code> . That is, the response is the grouping factor and the right hand side specifies the (non-factor) discriminators.
<code>data</code>	Data frame from which variables specified in formula are preferentially to be taken.
<code>method</code>	character, name of classification function (e.g. “ <code>lda</code> ”).
<code>x</code>	variable that should be plotted. See examples.
<code>col.wrong</code>	color to use for missclassified objects.
<code>ylim</code>	<code>ylim</code> for the plot.
<code>loo</code>	logical, whether leave-one-out estimate is used for prediction
<code>mfrow</code>	number of rows and columns in the graphics device, see par . If missing, number of rows equals number of classes, and 1 column.
<code>...</code>	further arguments passed to the underlying classification method or plot functions.

Value

The actual error rate.

Author(s)

Karsten Luebke

See Also

[partimat](#)

Examples

```

library(MASS)

# The name of the variable can be used for x
data(B3)
plineplot(PHASEN ~ ., data = B3, method = "lda",
          x = "EWAJW", xlab = "EWAJW")

# The plotted variable need not be in the data
data(iris)
iris2 <- iris[ , c(1,3,5)]
plineplot(Species ~ ., data = iris2, method = "lda",
          x = iris[ , 4], xlab = "Petal.Width")

```

plot.NaiveBayes *Naive Bayes Plot*

Description

Visualizes the marginal probabilities of predictor variables given the class.

Usage

```

## S3 method for class 'NaiveBayes':
plot(x, vars, n = 1000, legendplot = TRUE, lty, col,
     ylab = "Density", main = "Naive Bayes Plot", ...)

```

Arguments

x	an object of class NaiveBayes
vars	variables to be plotted. If missing, all predictor variables are plotted.
n	number of points used to plot the density line.
legendplot	logical, whether to print a legend
lty	line type for different classes, defaults to the first <code>length(x\$apriori)</code> colors of the current palette in use.
col	color for different classes, defaults to <code>rainbow(length(x\$apriori))</code> .
ylab	label for y-axis.
main	title of the plots.
...	further arguments passed to the underlying plot functions.

Details

For metric variables the estimated density is plotted. For categorical variables `mosaicplot` is called.

Author(s)

Karsten Luebke

See Also[NaiveBayes](#)**Examples**

```
data(iris)
mN <- NaiveBayes(Species ~ ., data = iris)
plot(mN)

mK <- NaiveBayes(Species ~ ., data = iris, usekernel = TRUE)
plot(mK)
```

predict.loclda *Localized Linear Discriminant Analysis (LocLDA)*

Description

Classifies new observations using parameters determined by the `loclda`-function.

Usage

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

Arguments

<code>object</code>	Object of class <code>loclda</code> .
<code>newdata</code>	Data frame of cases to be classified.
<code>...</code>	Further arguments are ignored.

Value

A list with components:

<code>class</code>	Vector (of class <code>factor</code>) of classifications.
<code>posterior</code>	Posterior probabilities for the classes. For details of computation see loclda (+ normalization so posterior-values add up to 1 for each observation).
<code>all.zero</code>	Vector (of class <code>integer</code>) indicating for which rows of <code>newdata</code> all corresponding posterior-values are $< 10^{-150}$ before normalization. Those observations are assigned to the class to whose (locally weighted) centroid they have the lowest euclidian distance.

Author(s)

Marc Zentgraf (marc-zentgraf@gmx.de) and Karsten Luebke (luebke@statistik.tu-dortmund.de)

See Also

[loclda](#)

Examples

```
data(B3)
x <- loclda(PHASEN ~ ., data = B3, subset = 1:80)
predict(x, B3[-(1:80),])
```

predict.locpvs *predict method for locpvs objects*

Description

Prediction of class membership and posterior probabilities in local models using pairwise variable selection.

Usage

```
## S3 method for class 'locpvs':
predict(object, newdata, quick = FALSE, return.subclass.prediction = TRUE, ...)
```

Arguments

object	an object of class ‘locpvs’, as that created by the function “ locpvs ”
newdata	a data frame or matrix containing new data. If not given the same data as used for training the ‘pvs’-model are used.
quick	indicator (logical), whether a quick, but less accurate computation of posterior probabilities should be used or not.
return.subclass.prediction	indicator (logical), whether the returned object includes posterior probabilities for each date in each subclass
...	Further arguments are passed to underlying predict calls.

Details

Posterior probabilities are predicted as if object is a standard ‘pvs’-model with the subclasses as classes. Then the posterior probabilities are summed over all subclasses for each class. The class with the highest value becomes the prediction.

If “quick=FALSE” the posterior probabilities for each case are computed using the pairwise coupling algorithm presented by Hastie, Tibshirani (1998). If “quick=TRUE” a much quicker solution is used, which leads to less accurate posterior probabilities. In almost all cases it doesn’t have a negative effect on the classification result.

Value

a list with components:

```
class          the predicted (upper) classes
posterior      posterior probabilities for the (upper) classes
subclass.posterioriors
                (only if "return.subclass.prediction=TRUE". A matrix containing
                posterior probabilities for the subclasses.
```

Author(s)

Gero Szepannek, <szepannek@statistik.tu-dortmund.de>, Christian Neumann

References

Szepannek, G. and Weihs, C. (2006) Local Modelling in Classification on Different Feature Subspaces. In *Advances in Data Mining.*, ed Perner, P., LNAI 4065, pp. 226-234. Springer, Heidelberg.

See Also

[locpvs](#) for learning 'locpvs'-models, [pvs](#) for pairwise variable selection without modeling subclasses, [predict.pvs](#) for predicting 'pvs'-models

Examples

```
## this example might be a bit artificial, but it sufficiently shows how locpvs has to be used

## learn a locpvs-model on the Vehicle dataset

library("mlbench")
data("Vehicle")

subclass <- Vehicle$class # use four car-types in dataset as subclasses
## aggregate "bus" and "van" to upper-class "big" and "saab" and "opel" to upper-class "small"
subclass_class <- matrix(c("bus", "van", "saab", "opel", "big", "big", "small", "small"), ncol=2)

## learn now a locpvs-model for the subclasses:
model <- locpvs(Vehicle[,1:18], subclass, subclass_class)
model # short summary, showing the class-pairs of the submodels
# together with the selected variables and the relation of sub- to upperclasses

## predict:
pred <- predict(model, Vehicle[,1:18])

## now you can look at the predicted classes:
pred$class
## or at the posterior probabilities:
pred$posterior
## or at the posterior probabilities for the subclasses:
pred$subclass.posterioriors
```

predict.meclight *Prediction of Minimal Error Classification*

Description

Classify multivariate observations in conjunction with `meclight` and `lda`.

Usage

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

Arguments

<code>object</code>	Object of class <code>meclight</code> .
<code>newdata</code>	Data frame of cases to be classified or, if <code>object</code> has a formula, a data frame with columns of the same names as the variables used. A vector will be interpreted as a row vector.
<code>...</code>	currently ignored

Details

Classify multivariate observations in conjunction with `meclight` and `lda`.

Value

<code>class</code>	The estimated class (<code>factor</code>).
<code>posterior</code>	Posterior probabilities for the classes.

Author(s)

Karsten Luebke

References

Roehl, M.C., Weihs, C., and Theis, W. (2002): Direct Minimization in Multivariate Classification. *Computational Statistics*, 17, 29-46.

See Also

`meclight`

Examples

```
data(iris)  
meclight.obj <- meclight(Species ~ ., data = iris)  
predict(meclight.obj, iris)
```

`predict.NaiveBayes` *Naive Bayes Classifier*

Description

Computes the conditional a-posterior probabilities of a categorical class variable given independent predictor variables using the Bayes rule.

Usage

```
## S3 method for class 'NaiveBayes':  
predict(object, newdata, threshold = 0.001, ...)
```

Arguments

<code>object</code>	An object of class "naiveBayes".
<code>newdata</code>	A dataframe with new predictors.
<code>threshold</code>	Value replacing cells with 0 probabilities.
<code>...</code>	passed to <code>dkernel</code> function if necessary.

Details

This implementation of Naive Bayes as well as this help is based on the code by David Meyer in the package `e1071` but extended for kernel estimated densities. The standard naive Bayes classifier (at least this implementation) assumes independence of the predictor variables. For attributes with missing values, the corresponding table entries are omitted for prediction.

Value

A list with the conditional a-posterior probabilities for each class and the estimated class are returned.

Author(s)

Karsten Luebke

See Also

[NaiveBayes](#), [dkernelnaiveBayes](#), [qda](#)

Examples

```
data(iris)  
m <- NaiveBayes(Species ~ ., data = iris)  
predict(m)
```

predict.pvs *predict method for pvs objects*

Description

Prediction of class membership and posterior probabilities using pairwise variable selection.

Usage

```
## S3 method for class 'pvs':
predict(object, newdata, quick = FALSE, detail = FALSE, ...)
```

Arguments

object	an object of class ‘pvs’, as that created by the function “ pvs ”
newdata	a data frame or matrix containing new data. If not given the same data as used for training the ‘pvs’-model are used.
quick	indicator (logical), whether a quick, but less accurate computation of posterior probabilities should be used or not.
detail	indicator (logical), whether the returned object includes additional information about the posterior probabilities for each date in each submodel.
...	Further arguments are passed to underlying predict calls.

Details

If “quick=FALSE” the posterior probabilities for each case are computed using the pairwise coupling algorithm presented by Hastie, Tibshirani (1998). If “quick=TRUE” a much quicker solution is used, which leads to less accurate posterior probabilities. In almost all cases it doesn’t have a negative effect on the classification result.

Value

a list with components:

class	the predicted classes
posterior	posterior probabilities for the classes
details	(only if “details=TRUE”. A list containing matrices of posterior probabilities computed by the classification method for each case and classpair.

Author(s)

Gero Szepannek, (szepannek@statistik.tu-dortmund.de), Christian Neumann

References

Szepannek, G. and Weihs, C. (2006) Variable Selection for Classification of More than Two Classes Where the Data are Sparse. In *From Data and Information Analysis to Knowledge Engineering.*, eds Spiliopoulou, M., Kruse, R., Borgelt, C., Nuernberger, A. and Gaul, W. pp. 700-708. Springer, Heidelberg.

See Also

[pvs](#)

Examples

```
## learn a pvs-model with half of the Satellite dataset,
## using "ks.test" as selection and "qda" as classification method:

library("mlbench")
data("Satellite")

model <- pvs(classes ~ ., Satellite[1:3218,], method = "qda", vs.method = "ks.test")
model # short summary, showing the class-pairs of the submodels and the selected variables

## now predict on the rest of the data set:

## pred <- predict(model, Satellite[3219:6435,]) # takes some time
pred <- predict(model, Satellite[3219:6435,], quick=TRUE) # that's much quicker

## now you can look at the predicted classes:
pred$class
## or the posterior probabilities:
pred$posterior
```

predict.sknn

Simple k Nearest Neighbours Classification

Description

Classifies new observations using the sknn learned by the sknn-function.

Usage

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

Arguments

object	Object of class sknn .
newdata	Data frame (or matrix) of cases to be classified.
...	...

Value

A list with elements 'class' and 'posterior'.

Author(s)

Karsten Luebke, (luebke@statistik.tu-dortmund.de)

See Also

[sknn](#), [knn](#)

Examples

```
data(iris)
x <- sknn(Species ~ ., data = iris)
predict(x, iris)
x <- sknn(Species ~ ., gamma = 10, kn = 10, data = iris)
predict(x, iris)
```

predict.svmlight *Interface to SVMlight*

Description

Predicts new observations using the SVM learned by the `svmlight`-function.

Usage

```
## S3 method for class 'svmlight':
predict(object, newdata, scal = TRUE, ...)
```

Arguments

<code>object</code>	Object of class <code>svmlight</code> .
<code>newdata</code>	Data frame (or matrix) of cases to be predicted.
<code>scal</code>	Logical, whether to scale membership values via <code>e.scal</code> .
<code>...</code>	...

Value

If a classification is learned (`type="C"`) in `svmlight` a list with elements 'class' and 'posterior' (scaled, if `scal = TRUE`).

If a Regression is learned (`type="R"`) in `svmlight` the predicted values.

Author(s)

Karsten Luebke, (luebke@statistik.tu-dortmund.de)

See Also

[svmlight](#), [svm](#)

Examples

```
## Not run:
data(iris)
x <- svmlight(Species ~ ., data = iris)
predict(x, iris)
## End(Not run)
```

pvs

Pairwise variable selection for classification

Description

Pairwise variable selection for numerical data, allowing the use of different classifiers and different variable selection methods.

Usage

```
pvs(x, ...)
```

```
## Default S3 method:
pvs(x, grouping, prior=NULL, method="lda",
     vs.method=c("ks.test", "stepclass", "greedy.wilks"), niveau=0.05,
     fold=10, impr=0.1, direct="backward", out=FALSE, ...)
```

```
## S3 method for class 'formula':
pvs(formula, data = NULL, ...)
```

Arguments

x	matrix or data frame containing the explanatory variables (required, if <code>formula</code> is not given). <code>x</code> must consist of numerical data only.
formula	A formula of the form <code>groups ~ x1 + x2 + ...</code> . That is, the response is the grouping factor (the classes) and the right hand side specifies the (numerical) discriminators. Interaction terms are not supported.
data	data matrix (rows=cases, columns=variables)
grouping	class indicator vector (a factor)
prior	prior probabilities for the classes. If not specified the prior probabilities will be set according to proportion in “grouping”. If specified the order of prior probabilities must be the same as in “grouping”.
method	character, name of classification function (e.g. “lda” (default)).
vs.method	character, name of variable selection method. Must be one of “ks.test” (default), “stepclass” or “greedy.wilks”.

niveau	used niveau for “ <code>ks.test</code> ”
fold	parameter for cross-validation, if “ <code>stepclass</code> ” is chosen ‘ <code>vs.method</code> ’
impr	least improvement of performance measure desired to include or exclude any variable (≤ 1), if “ <code>stepclass</code> ” is chosen ‘ <code>vs.method</code> ’
direct	direction of variable selection, if “ <code>stepclass</code> ” is chosen ‘ <code>vs.method</code> ’. Must be one of “ <code>forward</code> ”, “ <code>backward</code> ” (default) or “ <code>both</code> ”.
out	indicator (logical) for textoutput during computation (slows down computation!), if “ <code>stepclass</code> ” is chosen ‘ <code>vs.method</code> ’
...	further parameters passed to classification function (‘ <code>method</code> ’) or variable selection method (‘ <code>vs.method</code> ’)

Details

The classification “method” (e.g. ‘`lda`’) must have its own ‘`predict`’ method (like ‘`predict.lda`’ for ‘`lda`’) returns a list with an element ‘`posterior`’ containing the posterior probabilities. It must be able to deal with matrices as in `method(x, grouping, ...)`. Examples of such classification methods are ‘`lda`’, ‘`qda`’, ‘`rda`’, ‘`NaiveBayes`’ or ‘`sknn`’. For the classification methods “`svm`” and “`randomForest`” there are special routines implemented, to make them work with ‘`pvs`’ method even though their ‘`predict`’ methods don’t provide the demanded posteriors. However those two classifiers can not be used together with variable selection method “`stepclass`”.

‘`pvs`’ performs a variable selection using the selection method chosen in ‘`vs.method`’ for each pair of classes in ‘`x`’. Then for each pair of classes a submodel using ‘`method`’ is trained (using only the earlier selected variables for this class-pair).

If ‘`method`’ is “`ks.test`”, then for each variable the empirical distribution functions of the cases of both classes are compared via “`ks.test`”. Only variables with a p-values below ‘`niveau`’ are used for training the submodel for this pair of classes.

If ‘`method`’ is “`stepclass`” the variable selection is performed using the “`stepclass`” method.

If ‘`method`’ is “`greedy.wilks`” the variable selection is performed using Wilk’s lambda criterion.

Value

An object of class ‘`pvs`’ containing the following components:

<code>classes</code>	the classes in <code>grouping</code>
<code>prior</code>	used prior probabilities
<code>method</code>	name of used classification function
<code>vs.method</code>	name of used function for variable selection
<code>submodels</code>	containing a list of submodels. For each pair of classes there is a list element being another list of 3 containing the class-pair of this submodel, the selected variables for the subspace of classes and the result of the trained classification function.
<code>call</code>	the (matched) function call

Author(s)

Gero Szepannek, <szepannek@statistik.tu-dortmund.de>, Christian Neumann

References

Szepannek, G. and Weihs, C. (2006) Variable Selection for Classification of More than Two Classes Where the Data are Sparse. In *From Data and Information Analysis to Knowledge Engineering.*, eds Spiliopolou, M., Kruse, R., Borgelt, C., Nuernberger, A. and Gaul, W. pp. 700-708. Springer, Heidelberg.

Szepannek, G. (2008): *Different Subspace Classification - Datenanalyse, -interpretation, -visualisierung und Vorhersage in hochdimensionalen Raeumen*, ISBN 978-3-8364-6302-7, vdm, Saarbruecken.

See Also

[predict.pvs](#) for predicting 'pvs' models and [locpvs](#) for pairwisevariable selection in local models of several subclasses

Examples

```
## Example 1: learn an "lda" model on the waveform data using pairwise variable selection (pvs)
## and compare it to using lda without pvs

library(mlbench)
trainset <- mlbench.waveform(300)
pvsmodel <- pvs(trainset$x, trainset$classes, niveau=0.05) # default: using method="lda"
pvsmodel # short summary, showing the class-pairs of the submodels and the selected variables

testset <- mlbench.waveform(500)
## prediction of the test data set:
prediction <- predict(pvsmodel, testset$x)

## calculating the test error rate
1-sum(testset$classes==prediction$class)/length(testset$classes)
## Bayes error is 0.149

## comparison to performance of simple lda
ldamodel <- lda(trainset$x, trainset$classes)
LDAPrediction <- predict(ldamodel, testset$x)

## test error rate
1-sum(testset$classes==LDAPrediction$class)/length(testset$classes)

## Example 2: learn a "qda" model with pvs on half of the Satellite dataset, using "ks.test"

library("mlbench")
data("Satellite")

model <- pvs(classes ~ ., Satellite[1:3218,], method="qda", vs.method="ks.test")

## now predict on the rest of the data set:
```

```
## pred <- predict(model,Satellite[3219:6435,]) # takes some time
pred <- predict(model,Satellite[3219:6435,], quick=TRUE) # that's much quicker

## now you can look at the predicted classes:
pred$class
## or the posterior probabilities:
pred$posterior
```

quadplot

Plotting of 4 dimensional membership representation simplex

Description

For a 4 class discrimination problem the membership values of each class are visualized in a 3 dimensional barycentric coordinate system.

Usage

```
quadplot(e = NULL, f = NULL, g = NULL, h = NULL, angle = 75,
         scale.y = 0.6, label = 1:4, labelcol = rainbow(4),
         labelpch = 19, labelcex = 1.5, main = "", s3d.control = list(),
         simplex.control = list(), legend.control = list(), ...)
```

Arguments

e	either a matrix with 4 columns representing the membership values or a vector with the membership values of the first class
f	vector with the membership values of the second class
g	vector with the membership values of the third class
h	vector with the membership values of the fourth class
angle	angle between x and y axis
scale.y	scale of y axis related to x- and z axis
label	label for the classes
labelcol	colors to use for the labels
labelpch	pch for the labels
labelcex	cex for the labels
main	main title of the plot
s3d.control	a <i>list</i> with further arguments passed to the underlying <code>scatterplot3d</code> function call that sets up the plot
simplex.control	a <i>list</i> with further arguments passed to the underlying function call that draws the barycentric coordinate system

```

legend.control      a list with further arguments passed to the underlying function call that adds the
                    legend
...                further arguments passed to the underlying plot function that draws the data
                    points

```

Details

The membership values are calculated with `quadtrafo` and plotted with `scatterplot3d`.

Value

A `scatterplot3d` object.

Author(s)

Karsten Luebke, (luebke@statistik.tu-dortmund.de), and Uwe Ligges

References

Garczarek, Ursula Maria (2002): Classification rules in standardized partition spaces. Dissertation, University of Dortmund. URL <http://hdl.handle.net/2003/2789>

See Also

`triplot`, `scatterplot3d`

Examples

```

library("MASS")
data(B3)
opar <- par(mfrow = c(1, 2), pty = "s")
posterior <- predict(lda(PHASEN ~ ., data = B3))$post
s3d <- quadplot(posterior, col = rainbow(4)[B3$PHASEN],
               labelpch = 22:25, labelcex = 0.8,
               pch = (22:25)[apply(posterior, 1, which.max)],
               main = "LDA posterior assignments")
quadlines(centerlines(4), sp = s3d, lty = "dashed")

posterior <- predict(qda(PHASEN ~ ., data = B3))$post
s3d <- quadplot(posterior, col = rainbow(4)[B3$PHASEN],
               labelpch = 22:25, labelcex = 0.8,
               pch = (22:25)[apply(posterior, 1, which.max)],
               main = "QDA posterior assignments")
quadlines(centerlines(4), sp = s3d, lty = "dashed")
par(opar)

```

rda

*Regularized Discriminant Analysis (RDA)***Description**

Builds a classification rule using regularized group covariance matrices that are supposed to be more robust against multicollinearity in the data.

Usage

```
rda(x, ...)
```

Default S3 method:

```
rda(x, grouping = NULL, prior = NULL, gamma = NA,
    lambda = NA, regularization = c(gamma = gamma, lambda = lambda),
    crossval = TRUE, fold = 10, train.fraction = 0.5,
    estimate.error = TRUE, output = FALSE, startsimplex = NULL,
    max.iter = 100, trafo = TRUE, simAnn = FALSE, schedule = 2,
    T.start = 0.1, halflife = 50, zero.temp = 0.01, alpha = 2,
    K = 100, ...)
```

S3 method for class 'formula':

```
rda(formula, data, ...)
```

Arguments

<code>x</code>	Matrix or data frame containing the explanatory variables (required, if <code>formula</code> is not given).
<code>formula</code>	Formula of the form ‘groups ~ x1 + x2 + ...’.
<code>data</code>	A data frame (or matrix) containing the explanatory variables.
<code>grouping</code>	(Optional) a vector specifying the class for each observation; if not specified, the first column of ‘data’ is taken.
<code>prior</code>	(Optional) prior probabilities for the classes. Default: proportional to training sample sizes. “prior=1” indicates equally likely classes.
<code>gamma, lambda, regularization</code>	One or both of the rda-parameters may be fixed manually. Unspecified parameters are determined by minimizing the estimated error rate (see below).
<code>crossval</code>	Logical. If TRUE, in the optimization step the error rate is estimated by Cross-Validation, otherwise by drawing several training- and test-samples.
<code>fold</code>	The number of Cross-Validation- or Bootstrap-samples to be drawn.
<code>train.fraction</code>	In case of Bootstrapping: the fraction of the data to be used for training in each Bootstrap-sample; the remainder is used to estimate the misclassification rate.
<code>estimate.error</code>	Logical. If TRUE, the apparent error rate for the final parameter set is estimated.

<code>output</code>	Logical flag to indicate whether text output during computation is desired.
<code>startsimplex</code>	(Optional) a starting simplex for the Nelder-Mead-minimization.
<code>max.iter</code>	Maximum number of iterations for Nelder-Mead.
<code>trafo</code>	Logical; indicates whether minimization is carried out using transformed parameters.
<code>simAnn</code>	Logical; indicates whether Simulated Annealing shall be used.
<code>schedule</code>	Annealing schedule 1 or 2 (exponential or polynomial).
<code>T.start</code>	Starting temperature for Simulated Annealing.
<code>halflife</code>	Number of iterations until temperature is reduced to a half (schedule 1).
<code>zero.temp</code>	Temperature at which it is set to zero (schedule 1).
<code>alpha</code>	Power of temperature reduction (linear, quadratic, cubic,...) (schedule 2).
<code>K</code>	Number of iterations until temperature = 0 (schedule 2).
<code>...</code>	

Details

J.H. Friedman (see references below) suggested a method to fix almost singular covariance matrices in discriminant analysis. Basically, individual covariances as in QDA are used, but depending on two parameters (γ and λ), these can be shifted towards a diagonal matrix and/or the pooled covariance matrix. For ($\gamma = 0, \lambda = 0$) it equals QDA, for ($\gamma = 0, \lambda = 1$) it equals LDA.

You may fix these parameters at certain values or leave it to the function to try to find “optimal” values. If one parameter is given, the other one is determined using the R-function ‘`optimize`’. If no parameter is given, both are determined numerically by a Nelder-Mead-(Simplex-)algorithm with the option of using Simulated Annealing. The goal function to be minimized is the (estimated) misclassification rate; the misclassification rate is estimated either by Cross-Validation or by repeatedly dividing the data into training- and test-sets (Boostrapping).

Warning: If these sets are small, optimization is expected to produce almost random results. We recommend to adjust the parameters manually in such a case. In all other cases it is recommended to run the optimization several times in order to see whether stable results are gained.

Since the Nelder-Mead-algorithm is actually intended for *continuous* functions while the observed error rate by its nature is *discrete*, a greater number of Bootstrap-samples might improve the optimization by increasing the smoothness of the response surface (and, of course, by reducing variance and bias). If a set of parameters leads to singular covariance matrices, a penalty term is added to the misclassification rate which will hopefully help to maneuver back out of singularity (so do not worry about error rates greater than one during optimization).

Value

A list of class `rda` containing the following components:

<code>call</code>	The (matched) function call.
<code>regularization</code>	vector containing the two regularization parameters (<code>gamma</code> , <code>lambda</code>)
<code>classes</code>	the names of the classes

prior	the prior probabilities for the classes
error.rate	apparent error rate (if computation was not suppressed), and, if any optimization took place, the final (cross-validated or bootstrapped) error rate estimate as well.
means	Group means.
covariances	Array of group covariances.
covpooled	Pooled covariance.
converged	(Logical) indicator of convergence (only for Nelder-Mead).
iter	Number of iterations actually performed (only for Nelder-Mead).

More details

The explicit definition of γ , λ and the resulting covariance estimates is as follows:

The pooled covariance estimate $\hat{\Sigma}$ is given as well as the individual covariance estimates $\hat{\Sigma}_k$ for each group.

First, using λ , a convex combination of these two is computed:

$$\hat{\Sigma}_k(\lambda) := (1 - \lambda)\hat{\Sigma}_k + \lambda\hat{\Sigma}.$$

Then, another convex combination is constructed using the above estimate and a (scaled) identity matrix:

$$\hat{\Sigma}_k(\lambda, \gamma) = (1 - \gamma)\hat{\Sigma}_k(\lambda) + \gamma\frac{1}{d}\text{tr}[\hat{\Sigma}_k(\lambda)]\mathbf{I}.$$

The factor $\frac{1}{d}\text{tr}[\hat{\Sigma}_k(\lambda)]$ in front of the identity matrix \mathbf{I} is the mean of the diagonal elements of $\hat{\Sigma}_k(\lambda)$, so it is the mean variance of all d variables assuming the group covariance $\hat{\Sigma}_k(\lambda)$.

For the four extremes of (γ, λ) the covariance structure reduces to special cases:

- $(\gamma = 0, \lambda = 0)$: QDA - individual covariance for each group.
- $(\gamma = 0, \lambda = 1)$: LDA - a common covariance matrix.
- $(\gamma = 1, \lambda = 0)$: Conditional independent variables - similar to Naive Bayes, but variable variances within group (main diagonal elements) are equal.
- $(\gamma = 1, \lambda = 1)$: Classification using euclidean distance - as in previous case, but variances are the same for all groups. Objects are assigned to group with nearest mean.

Author(s)

Christian Röver, <roever@statistik.tu-dortmund.de>

References

Friedman, J.H. (1989): Regularized Discriminant Analysis. In: *Journal of the American Statistical Association* 84, 165-175.

Press, W.H., Flannery, B.P., Teukolsky, S.A., Vetterling, W.T. (1992): *Numerical Recipes in C*. Cambridge: Cambridge University Press.

See Also

[predict.rda](#), [lda](#), [qda](#)

Examples

```
data(iris)
x <- rda(Species ~ ., data = iris, gamma = 0.05, lambda = 0.2)
predict(x, iris)
```

shardsplot

*Plotting Eight Direction Arranged Maps or Self-Organizing Maps***Description**

Plotting method for objects of class `EDAM` or `som`.

Usage

```
shardsplot(object, plot.type = c("eight", "four", "points", "n"),
  expand = 1, stck = TRUE, grd = FALSE, standardize = FALSE,
  data.or = NA, label = FALSE, plot = TRUE, classes = 0,
  vertices = TRUE, classcolors = "rainbow", wghts = 0,
  xlab = "Dimension 1", ylab = "Dimension 2", xaxs = "i",
  yaxs = "i", plot.data.column = NA,
  log.classes = FALSE, revert.colors = FALSE, ...)

level_shardsplot(object, par.names, rows = 1:NCOL(object$data),
  centers = rep(NA, length(par.names)), class.labels = NA,
  revert.colors = rep(FALSE, length(par.names)),
  log.classes = rep(FALSE, length(par.names)),
  centeredcolors = colorRamp(c("red", "white", "blue")),
  mfrow = c(2, 2), plot.type = c("eight", "four", "points", "n"),
  expand = 1, stck = TRUE, grd = FALSE, standardize = FALSE,
  label = FALSE, plot = TRUE, vertices = TRUE, classcolors = "topo",
  wghts = 0, xlab = "Dimension 1", ylab = "Dimension 2",
  xaxs = "i", yaxs = "i", ...)

## S3 method for class 'EDAM':
plot(...)
```

Arguments

<code>object</code>	an object of class <code>EDAM</code> or <code>som</code> .
<code>par.names</code>	names used to lable the data columns
<code>rows</code>	vector with indices of colomns to be plotted
<code>centers</code>	vector of type numeric defining the class centers for the data. NA if data does not have a center.
<code>class.labels</code>	matrix of type text and dimension $(3, NROW(object$data))$ defining the lables to be used for maximum, minimum and central value.

<code>centeredcolors</code>	colors to represent the classes with a central value
<code>mfrow</code>	parameter defining number of plots on a page. see par
<code>plot.type</code>	a character giving the shape of the shards. Available are “eight” and “four” for octagons resp. rectangles, and “points” for points. If <code>plot.type</code> is “n”, no shards are plotted at all.
<code>expand</code>	a numeric giving the relative expansion of the axes. A value greater than one implies smaller shards. Varying <code>expand</code> can be sensible for visual reasons.
<code>stck</code>	logical. If TRUE the cells are varied continuously corresponding to the differences of direct neighbors in the origin space. Within this variation the relative order of the cells is always preserved.
<code>grd</code>	logical. If TRUE (which automatically sets <code>stck</code> to TRUE), the variation of cells is restricted to their original discrete values.
<code>standardize</code>	logical. If TRUE, then the measurements in <code>object\$preimages</code> are standardized before calculating Euclidean distances. Measurements are standardized for each variable by dividing by the variable’s standard deviation. Meaningless if <code>object\$preimages</code> is a dissimilarity matrix.
<code>data.or</code>	original data and classes where the first <code>k</code> columns are variables and the <code>(k+1)</code> -th column are the classes. If defined and class of <code>object</code> is <code>som</code> , <code>data.or</code> is used to assign a class to each codebook. There a codebook receives the class, from which the majority of its assigned objects origins.
<code>label</code>	logical. If TRUE, the shards are labeled by the rownames of the preimages.
<code>plot</code>	logical. If FALSE, all graphical output is suppressed.
<code>classes</code>	a vector giving alternative classes for objects of class EDAM; <code>classes</code> have to be given in the original order of the data to which EDAM was applied.
<code>vertices</code>	logical. If TRUE the grid is drawn.
<code>classcolors</code>	colors to represent the classes, or a character giving the <i>colorscale</i> for the classes. Since now available scales are <code>rainbow</code> , <code>topo</code> and <code>gray</code> .
<code>wghts</code>	an optional vector of length <code>k</code> giving relative weights of the variables in computing Euclidean distances. Meaningless if <code>object\$preimages</code> is a dissimilarity matrix.
<code>xaxs</code>	see par
<code>yaxs</code>	see par
<code>xlab</code>	see par
<code>ylab</code>	see par
<code>...</code>	further plotting parameters.
<code>plot.data.column</code>	column index defining from <code>data.or</code> providing the data used to calculate the coloring of the cells.
<code>log.classes</code>	boolean indicating that the data should be transformed with the logarithmic function before calculating the cell coloring
<code>revert.colors</code>	boolean indicating that the colorscale should be reverted.

Details

`level_shardsplot` uses multiple `shardsplot` representations of a SOM in order to depict how the data used to calculate the SOM is distributed across the map. Two representations are possible for the data, first with a single color ramp from the minimum value to the maximum value. The second representation is useful for data for which a basic value exists somewhere between minimum and maximum for which a special color representation should be used (e.g. 0 is indicated with white).

If `plot.type` is “four” or “eight”, the shape of each shard depends on the relative distances of the actual object or codebook to its up to eight neighbours. If `plot.type` is “eight”, `shardsplot` corresponds to the representation method suggested by Cottrell and de Bodt (1996) for Kohonen Self-Organizing Maps. If `plot.type` is “points”, `shardsplot` reduces to a usual scatter plot.

Value

The following list is (invisibly) returned:

<code>Cells.ex</code>	the images of the visualized data
<code>S</code>	the criterion of the visualization

Author(s)

Nils Raabe, `level_shardsplot` function from Dominik Reusser

References

Cottrell, M., and de Bodt, E. (1996). A Kohonen Map Representation to Avoid Misleading Interpretations. *Proceedings of the European Symposium on Artificial Neural Networks, D-Facto*, pp. 103–110.

See Also

[EDAM](#), [TopoS](#), [som](#)

Examples

```
# Compute clusters and an Eight Directions Arranged Map for the
# country data. Plotting the result.
data(countries)
logcount <- log(countries[,2:7])
sdlogcount <- apply(logcount, 2, sd)
logstand <- t((t(logcount) / sdlogcount) * c(1,2,6,5,5,3))
cclasses <- cutree(hclust(dist(logstand)), k = 6)
countryEDAM <- EDAM(logstand, classes = cclasses, sa = FALSE,
  iter.max = 10, random = FALSE)
plot(countryEDAM, vertices = FALSE, label = TRUE, stck = FALSE)

# Compute and plot a Self-Organizing Map for the iris data
data(iris)
library(som)
```

```

irissom <- som(iris[,1:4], xdim = 6, ydim = 14)
shardsplot(irissom, data.or = iris, vertices = FALSE)
opar <- par(xpd = NA)
legend(7.5, 6.1, col = rainbow(3), xjust = 0.5, yjust = 0,
       legend = levels(iris[, 5]), pch = 16, horiz = TRUE)
par(opar)

level_shardsplot(irissom, par.names = names(iris),
                 class.labels = NA, mfrow = c(2,2))

```

sknn

Simple k nearest Neighbours

Description

Function for simple knn classification.

Usage

```

sknn(x, ...)

## Default S3 method:
sknn(x, grouping, kn = 3, gamma=0, ...)
## S3 method for class 'data.frame':
sknn(x, ...)
## S3 method for class 'matrix':
sknn(x, grouping, ..., subset, na.action = na.fail)
## S3 method for class 'formula':
sknn(formula, data = NULL, ..., subset, na.action = na.fail)

```

Arguments

x	matrix or data frame containing the explanatory variables (required, if <i>formula</i> is not given).
grouping	factor specifying the class for each observation (required, if <i>formula</i> is not given).
formula	formula of the form <code>groups ~ x1 + x2 + ...</code> . That is, the response is the grouping factor and the right hand side specifies the (non-factor) discriminators.
data	Data frame from which variables specified in <i>formula</i> are preferentially to be taken.
kn	Number of nearest neighbours to use.
gamma	gamma parameter for rbf in knn. If <code>gamma=0</code> ordinary knn classification is used.
subset	An index vector specifying the cases to be used in the training sample. (Note: If given, this argument must be named.)

`na.action` specify the action to be taken if NAs are found. The default action is for the procedure to fail. An alternative is `na.omit`, which leads to rejection of cases with missing values on any required variable. (Note: If given, this argument must be named.)

...

Details

If `gamma > 0` an gaussian like density is used to weight the classes of the `kn` nearest neighbors. `weight = exp(-gamma * distance)`. This is similar to an rbf kernel. If the distances are large it may be useful to `scale` the data first.

Value

A list containing the function call.

Author(s)

Karsten Luebke, (luebke@statistik.tu-dortmund.de)

See Also

`predict.sknn`, `knn`

Examples

```
data(iris)
x <- sknn(Species ~ ., data = iris)
x <- sknn(Species ~ ., gamma = 4, data = iris)
```

stepclass

Stepwise variable selection for classification

Description

Forward/backward variable selection for classification using any specified classification function and selecting by estimated classification performance measure from `ucpm`.

Usage

```
stepclass(x, ...)

## Default S3 method:
stepclass(x, grouping, method, improvement = 0.05, maxvar = Inf,
          start.vars = NULL, direction = c("both", "forward", "backward"),
          criterion = "CR", fold = 10, cv.groups = NULL, output = TRUE,
          minlvar = TRUE, ...)
## S3 method for class 'formula':
stepclass(formula, data, method, ...)
```

Arguments

<code>x</code>	matrix or data frame containing the explanatory variables (required, if <code>formula</code> is not given).
<code>formula</code>	A formula of the form <code>groups ~ x1 + x2 + ...</code> . That is, the response is the grouping factor and the right hand side specifies the (non-factor) discriminators. Interaction terms are not supported.
<code>data</code>	data matrix (rows=cases, columns=variables)
<code>grouping</code>	class indicator vector (a factor)
<code>method</code>	character, name of classification function (e.g. <code>"lda"</code>).
<code>improvement</code>	least improvement of performance measure desired to include or exclude any variable (≤ 1)
<code>maxvar</code>	maximum number of variables in model
<code>start.vars</code>	set variables to start with (indices or names). Default is no variables if <code>'direction'</code> is <code>"forward"</code> or <code>"both"</code> , and all variables if <code>'direction'</code> is <code>"backward"</code> .
<code>direction</code>	<code>"forward"</code> , <code>"backward"</code> or <code>"both"</code> (default)
<code>criterion</code>	performance measure taken from <code>ucpm</code> .
<code>fold</code>	parameter for cross-validation; omitted if <code>'cv.groups'</code> is specified.
<code>cv.groups</code>	vector of group indicators for cross-validation. By default assigned automatically.
<code>output</code>	indicator (logical) for textoutput during computation (slows down computation!)
<code>minlvar</code>	logical, whether to include at least one variable in the model, even if the prior itself already is a reasonable model.
<code>...</code>	further parameters passed to classification function (<code>'method'</code>), e.g. priors etc.

Details

The classification "method" (e.g. `'lda'`) must have its own `'predict'` method (like `'predict.lda'` for `'lda'`) that either returns a matrix of posterior probabilities or a list with an element `'posterior'` containing that matrix instead. It must be able to deal with matrices as in `method(x, grouping, ...)`

Then a stepwise variable selection is performed. The initial model is defined by the provided starting variables; in every step new models are generated by including every single variable that is not in the model, and by excluding every single variable that is in the model. The resulting performance measure for these models are estimated (by cross-validation), and if the maximum value of the chosen criterion is better than `'improvement'` plus the value so far, the corresponding variable is in- or excluded. The procedure stops, if the new best value is not good enough, or if the specified maximum number of variables is reached.

If `'direction'` is `"forward"`, the model is only extended (by including further variables), if `'direction'` is `"backward"`, the model is only reduced (by excluding variables from the model).

Value

An object of class ‘stepclass’ containing the following components:

call	the (matched) function call.
method	name of classification function used (e.g. “lda”).
start.variables	vector of starting variables.
process	data frame showing selection process (included/excluded variables and performance measure).
model	the final model: data frame with 2 columns; indices and names of variables.
performance.measure	value of the criterion used by ucpm
formula	formula of the form ‘response ~ list + of + selected + variables’

Author(s)

Christian Röver, (roever@statistik.tu-dortmund.de), Irina Czogiel

See Also

[step](#), [stepAIC](#), and [greedy.wilks](#) for stepwise variable selection according to Wilk’s lambda

Examples

```
data(iris)
library(MASS)
iris.d <- iris[,1:4] # the data
iris.c <- iris[,5]  # the classes
sc_obj <- stepclass(iris.d, iris.c, "lda", start.vars = "Sepal.Width")
sc_obj
plot(sc_obj)

## or using formulas:
sc_obj <- stepclass(Species ~ ., data = iris, method = "qda",
  start.vars = "Sepal.Width", criterion = "AS") # same as above
sc_obj
## now you can say stuff like
## qda(sc_obj$formula, data = B3)
```

 svmlight

Interface to SVMlight

Description

Function to call SVMlight from R for classification. Multiple group classification is done with the one-against-rest partition of data.

Usage

```
svmlight(x, ...)

## Default S3 method:
svmlight(x, grouping, temp.dir = NULL, pathsvm = NULL,
         del = TRUE, type = "C", class.type = "oaa", svm.options = NULL,
         prior = NULL, out = FALSE, ...)
## S3 method for class 'data.frame':
svmlight(x, ...)
## S3 method for class 'matrix':
svmlight(x, grouping, ..., subset, na.action = na.fail)
## S3 method for class 'formula':
svmlight(formula, data = NULL, ..., subset,
         na.action = na.fail)
```

Arguments

<code>x</code>	matrix or data frame containing the explanatory variables (required, if <code>formula</code> is not given).
<code>grouping</code>	factor specifying the class for each observation (required, if <code>formula</code> is not given).
<code>formula</code>	formula of the form <code>groups ~ x1 + x2 + ...</code> . That is, the response is the grouping factor and the right hand side specifies the (non-factor) discriminators.
<code>data</code>	Data frame from which variables specified in <code>formula</code> are preferentially to be taken.
<code>temp.dir</code>	directory for temporary files.
<code>pathsvm</code>	Path to SVMlight binaries (required, if path is unknown by the OS).
<code>del</code>	Logical: whether to delete temporary files
<code>type</code>	Perform "C"=Classification or "R"=Regression
<code>class.type</code>	Multiclass scheme to use. See details.
<code>svm.options</code>	Optional parameters to SVMlight. For further details see: "How to use" on http://svmlight.joachims.org/ .
<code>prior</code>	A Priori probabilities of classes.
<code>out</code>	Logical: whether SVMlight output should be printed on console (only for Windows OS.)
<code>subset</code>	An index vector specifying the cases to be used in the training sample. (Note: If given, this argument must be named.)
<code>na.action</code>	specify the action to be taken if NAs are found. The default action is for the procedure to fail. An alternative is <code>na.omit</code> , which leads to rejection of cases with missing values on any required variable. (Note: If given, this argument must be named.)
<code>...</code>	

Details

Function to call SVMlight from R for classification (`type="C"`). SVMlight is an implementation of Vapnik's Support Vector Machine. It is written in C by Thorsten Joachims. On the homepage (see below) the source-code and several binaries for SVMlight are available. If more than two classes are given the SVM is learned by the one-against-all scheme (`class.type="oaa"`). That means that each class is trained against the other $K-1$ classes. The class with the highest decision function in the SVM wins. So K SVMs have to be learned. If `class.type="oao"` each class is tested against every other and the final class is elected by a majority vote.

If `type="R"` a SVM Regression is performed.

Value

A list containing the function call and the result of SVMlight.

Requirements

SVMlight (<http://svmlight.joachims.org/>) must be installed before using this interface.

Author(s)

Karsten Luebke, (luebke@statistik.tu-dortmund.de), Andrea Preusser

References

<http://svmlight.joachims.org/>

See Also

[predict.svmlight](#), [svm](#),

Examples

```
## Not run:
## Only works if the svmlight binaries are in the path.
data(iris)
x <- svmlight(Species ~ ., data = iris)
## Using RBF-Kernel with gamma=0.1:
data(B3)
x <- svmlight(PHASEN ~ ., data = B3, svm.options = "-t 2 -g 0.1")
## End(Not run)
```

triframe	<i>Barycentric plots</i>
----------	--------------------------

Description

Function to add a frame to an existing (barycentric) plot.

Usage

```
triframe(label = 1:3, label.col = 1, cex = 1, ...)
```

Arguments

label	labels for the three corners of the plot.
label.col	text color for labels.
cex	Magnification factor for label text relative to the default.
...	Further graphical parameters passed to trilines .

Author(s)

Christian Röver, <roever@statistik.tu-dortmund.de>

See Also

[triplot](#), [trilines](#), [trigrid](#), [centerlines](#)

Examples

```
triplot(grid = TRUE, frame = FALSE)      # plot without frame
some.triangle <- rbind(c(0, 0.65, 0.35), c(0.53, 0.47, 0),
                      c(0.72, 0, 0.28))[c(1:3, 1), ]
trilines(some.triangle, col = "red", pch = 16, type = "b")
triframe(label = c("left", "top", "right"), col = "blue",
         label.col = "green3")          # frame on top of points
```

trigrid	<i>Barycentric plots</i>
---------	--------------------------

Description

Function to add a grid to an existing (barycentric) plot.

Usage

```
trigrid(x = seq(0.1, 0.9, by = 0.1), y = NULL, z = NULL,
       lty = "dashed", col = "grey", ...)
```

Arguments

x	Values along which to draw grid lines for first dimension (or all dimensions if y and z omitted). For NO grid lines in some dimensions just supply an NA.
y	Grid lines for second dimension.
z	Grid lines for third dimension.
lty	Line type (see par).
col	Line colour (see par).
...	Further graphical parameters passed to trilines .

Details

Grid lines illustrate the set of points for which one of the dimensions is held constant; e.g. horizontal lines contain all points with a certain value y for the second dimension, connecting the two extreme points (0,y,1-y) and (1-y,y,0).

Grids may be designed more flexible than with `triplot`'s `grid` option.

Author(s)

Christian Röver, roever@statistik.tu-dortmund.de

See Also

[triplot](#), [trilines](#), [triframe](#), [centerlines](#)

Examples

```
triplot(grid = FALSE)
trigrd(c(1/3, 0.5)) # same grid for all 3 dimensions

triplot(grid = c(1/3, 0.5)) # (same effect)

triplot(grid = FALSE)
# different grids for all dimensions:
trigrd(x = 1/3, y = 0.5, z = seq(0.2, 0.8, by=0.2))

triplot(grid = FALSE)
# grid for third dimension only:
trigrd(x = NA, y = NA, z = c(0.1, 0.2, 0.4, 0.8))
```

triperplines *Barycentric plots*

Description

Function to add a point and the corresponding perpendicular lines to all three sides to an existing (barycentric) plot.

Usage

```
triperplines(x, y = NULL, z = NULL, lcol = "red", pch = 17, ...)
```

Arguments

x	fraction of first component OR 3-element vector (for all three components, omitting y and z).
y	(optional) fraction of second component.
z	(optional) fraction of third component.
lcol	line color
pch	plotting character. pch = 0 for no point
...	Further graphical parameters (see points , lines and par).

Details

Adds a (single!) point and lines to an existing plot (generated by [triplot](#)). The lines originate from the point and run (perpendicular) towards all three sides. The lengths (and proportions) of these lines are identical to those of x, y and z.

Value

a 2-column-matrix containing plot coordinates.

Author(s)

Christian Röver, <roever@statistik.tu-dortmund.de>

See Also

[triplot](#), [tripoints](#), [trilines](#), [tritrafo](#)

Examples

```
triplot() # empty plot
triperplines(1/2, 1/3, 1/6)
```

triplot

*Barycentric plots***Description**

Function to produce triangular (barycentric) plots illustrating proportions of 3 components, e.g. discrete 3D-distributions or mixture fractions that sum up to 1.

Usage

```
triplot(x = NULL, y = NULL, z = NULL, main = "", frame = TRUE,
        label = 1:3, grid = seq(0.1, 0.9, by = 0.1), center = FALSE,
        set.par = TRUE, ...)
```

Arguments

<code>x</code>	Vector of fractions of first component OR 3-column matrix containing all three components (omitting <code>y</code> and <code>z</code>) OR 3-element vector (for all three components, omitting <code>y</code> and <code>z</code>).
<code>y</code>	(Optional) vector of fractions of second component.
<code>z</code>	(Optional) vector of fractions of third component.
<code>main</code>	Main title
<code>frame</code>	Controls whether a frame (triangle) and labels are drawn.
<code>label</code>	(Character) vector of labels for the three corners.
<code>grid</code>	Values along which grid lines are to be drawn (or <code>FALSE</code> for no grid at all). Default is steps of 10 percent.
<code>center</code>	Controls whether or not to draw centerlines at which there is a 'tie' between any two dimensions (see also centerlines).
<code>set.par</code>	Controls whether graphical parameter <code>mar</code> is set so the plot fills the window (see par).
<code>...</code>	Further graphical parameters passed to trilines .

Details

The barycentric plot illustrates the set of points (x,y,z) with x,y,z between 0 and 1 and $x+y+z=1$; that is, the triangle spanned by $(1,0,0)$, $(0,1,0)$ and $(0,0,1)$ in 3-dimensional space. The three dimensions x , y and z correspond to lower left, upper and lower right corner of the plot. The greater the share of x in the proportion, the closer the point is to the lower left corner; Points on the opposite (upper right) side have a zero x -fraction. The grid lines show the points at which one dimension is held constant, horizontal lines for example contain points with a constant second dimension.

Author(s)

Christian Röver, (roever@statistik.tu-dortmund.de)

See Also

[tripoints](#), [trilines](#), [triperplines](#), [trigrd](#), [triframe](#) for points, lines and layout, [tritrafo](#) for placing labels, and [quadplot](#) for the same in 4 dimensions.

Examples

```
# illustrating probabilities:
tripplot(label = c("1, 2 or 3", "4 or 5", "6"),
         main = "die rolls: probabilities", pch = 17)
triperplines(1/2, 1/3, 1/6)

# expected...
tripplot(1/2, 1/3, 1/6, label = c("1, 2 or 3", "4 or 5", "6"),
         main = "die rolls: expected and observed frequencies", pch = 17)
# ... and observed frequencies.
dierolls <- matrix(sample(1:3, size = 50*20, prob = c(1/2, 1/3, 1/6),
                        replace = TRUE), ncol = 50)
frequencies <- t(apply(dierolls, 1,
                      function(x) (summary(factor(x, levels = 1:3)))) / 50)
tripoints(frequencies)

# LDA classification posterior:
data(iris)
require(MASS)
pred <- predict(lda(Species ~ ., data = iris),iris)
plotchar <- rep(1,150)
plotchar[pred$class != iris$Species] <- 19
tripplot(pred$posterior, label = colnames(pred$posterior),
         main = "LDA posterior assignments", center = TRUE,
         pch = plotchar, col = rep(c("blue", "green3", "red"), rep(50, 3)),
         grid = TRUE)
legend(x = -0.6, y = 0.7, col = c("blue", "green3", "red"),
      pch = 15, legend = colnames(pred$posterior))
```

tripoints

Barycentric plots

Description

Function to add points or lines to an existing (barycentric) plot.

Usage

```
tripoints(x, y = NULL, z = NULL, ...)
trilines(x, y = NULL, z = NULL, ...)
```

Arguments

x	Vector of fractions of first component OR 3-column matrix containing all three components (omitting y and z) OR 3-element vector (for all three components, omitting y and z).
y	(optional) vector of fractions of second component.
z	(optional) vector of fractions of third component.
...	Further graphical parameters (see points and par).

Details

Adds points or lines to an existing plot (generated by [triplot](#)).

Author(s)

Christian Röver, roever@statistik.tu-dortmund.de

See Also

[points](#), [lines](#), [triplot](#), [tritrafo](#), [centerlines](#)

Examples

```
triplot() # empty plot
tripoints(0.1, 0.2, 0.7) # a point
tripoints(c(0.2, 0.6), c(0.3, 0.3), c(0.5, 0.1),
  pch = c(2, 6)) # two points
trilines(c(0.1, 0.6), c(0.2, 0.3), c(0.7, 0.1),
  col = "blue", lty = "dotted") # a line

trilines(centerlines(3))
```

tritrafo

Barycentric plots

Description

Function to carry out the transformation into 2D space for [triplot](#), [trilines](#) etc.

Usage

```
tritrafo(x, y = NULL, z = NULL, check = TRUE, tolerance = 0.0001)
```

Arguments

x	Vector of fractions of first component OR 3-column matrix containing all three components (omitting y and z) OR 3-element vector (for all three components, omitting y and z).
y	(optional) vector of fractions of second component.
z	(optional) vector of fractions of third component.
check	if TRUE, it is checked whether $x+y+z=1$ and $x, y, z \geq 0$ for all cases.
tolerance	tolerance for above sum check.

Details

Projects the mixture given by x, y, and z with x, y, z between one and zero and $x+y+z=1$ into a two-dimensional space.

For further details see [triplot](#).

Value

A matrix with two columns corresponding to the two dimensions.

Author(s)

Christian Röver, (roever@statistik.tu-dortmund.de)

See Also

[triplot](#), [tripoints](#), [trilines](#), [trigrd](#)

Examples

```
tritrafo(0.1, 0.2, 0.7)
tritrafo(0.1, 0.2, 0.6) # warning

triplot()
points(tritrafo(0.1, 0.2, 0.7), col="red")
tripoints(0.1, 0.2, 0.7, col="green") # the same

tritrafo(c(0.1,0.2), c(0.3,0.4), c(0.6,0.4))
tritrafo(diag(3))

point <- c(0.25,0.6,0.15)
triplot(point, pch=16)
text(tritrafo(point), "(0.25, 0.60, 0.15)", adj=c(0.5,2)) # add a label
```

ucpm

*Uschi's classification performance measures***Description**

Function to calculate the Correctness Rate, the Accuracy, the Ability to Seperate and the Confidence of a classification rule.

Usage

```
ucpm(m, tc, ec = NULL)
```

Arguments

m	matrix of (scaled) membership values
tc	vector of true classes
ec	vector of estimated classes (only required if scaled membership values are used)

Details

- The *correctness rate* is the estimator for the correctness of a classification rule (1-error rate).
- The *accuracy* is based on the euclidean distances between (scaled) membership vectors and the vectors representing the true class corner. These distances are standardized so that a measure of 1 is achieved if all vectors lie in the correct corners and 0 if they all lie in the center.
- Analogously, the *ability to seperate* is based on the distances between (scaled) membership vectors and the vector representing the corresponding assigned class corner.
- The *confidence* is the mean of the membership values of the assigned classes.

Value

A list with elements:

CR	Correctness Rate
AC	Accuracy
AS	Ability to Seperate
CF	Confidence
CFvec	Confidence for each (true) class

Author(s)

Karsten Luebke, {luebke@statistik.tu-dortmund.de}

References

Garczarek, Ursula Maria (2002): Classification rules in standardized partition spaces. Dissertation, University of Dortmund. URL <http://hdl.handle.net/2003/2789>

Examples

```
library(MASS)
data(iris)
ucpm(predict(lda(Species ~ ., data = iris))$posterior, iris$Species)
```

Index

*Topic **aplot**

- triframe, 60
- trigrid, 60
- triperplines, 62
- triplot, 63
- tripoints, 64

*Topic **attribute**

- corclust, 8

*Topic **category**

- NaiveBayes, 28
- predict.NaiveBayes, 39

*Topic **classif**

- b.scal, 2
- benchB3, 4
- betascale, 5
- centerlines, 7
- classscatter, 7
- corclust, 8
- drawparti, 11
- e.scal, 13
- hmm.sop, 21
- locpvs, 24
- meclight.default, 26
- NaiveBayes, 28
- nm, 29
- partimat, 31
- plineplot, 33
- plot.NaiveBayes, 34
- predict.locpvs, 36
- predict.meclight, 38
- predict.NaiveBayes, 39
- predict.pvs, 40
- predict.sknn, 41
- predict.svmlight, 42
- pvs, 43
- quadplot, 46
- sknn, 54
- svmlight, 57
- ucpm, 67

*Topic **cluster**

- corclust, 8

*Topic **datasets**

- B3, 3
- countries, 10

*Topic **distribution**

- dkernel, 10

*Topic **dplot**

- centerlines, 7
- classscatter, 7
- drawparti, 11
- partimat, 31
- plineplot, 33
- plot.NaiveBayes, 34
- quadplot, 46
- tritrafo, 65

*Topic **hplot**

- shardsplot, 51

*Topic **manip**

- corclust, 8

*Topic **multivariate**

- corclust, 8
- EDAM, 14
- errormatrix, 16
- friedman.data, 18
- greedy.wilks, 19
- loclda, 22
- predict.loclda, 35
- pvs, 43
- rda, 48
- stepclass, 55

*Topic **nonparametric**

- dkernel, 10

*Topic **ts**

- calc.trans, 6
- hmm.sop, 21

- b.scal, 2, 5, 6
- B3, 3, 4, 5
- benchB3, 4, 4

- betascale, 2, 3, 5
- calc.trans, 6, 21
- centerlines, 7, 60, 61, 63, 65
- classscatter, 7
- contour, 12
- corclust, 8
- countries, 10
- density, 11, 28, 29
- dkernel, 10, 39
- drawpart, 11, 32
- e.scal, 3, 6, 13, 42
- EDAM, 14, 51–53
- errormatrix, 16
- friedman.data, 18
- greedy.wilks, 19, 25, 43, 44, 57
- hclust, 9
- hmm.sop, 21
- image, 12
- knn, 30, 42, 55
- ks.test, 25, 43, 44
- lda, 2, 8, 12, 18, 24, 25, 27, 31, 33, 38, 43, 44, 50, 56
- legend, 34, 47
- level_shardsplot (*shardsplot*), 51
- lines, 62, 65
- loclda, 22, 35, 36
- locpvs, 24, 36, 37, 45
- manova, 20
- meclight, 38
- meclight (*meclight.default*), 26
- meclight.default, 26
- mosaicplot, 34
- na.omit, 30, 32, 55, 58
- NaiveBayes, 11, 28, 34, 35, 39, 44
- naiveBayes, 12, 29, 31, 39
- nm, 29
- optim, 27
- optimize, 49
- par, 33, 52, 61–63, 65
- partimat, 12, 31, 33
- pbeta, 2
- plineplot, 33
- plot, 8
- plot.EDAM (*shardsplot*), 51
- plot.NaiveBayes, 29, 34
- plot.rda (*rda*), 48
- plot.stepclass (*stepclass*), 55
- points, 62, 65
- predict.lda, 44, 56
- predict.loclda, 23, 24, 35
- predict.locpvs, 26, 36
- predict.meclight, 28, 38
- predict.NaiveBayes, 29, 39
- predict.pvs, 37, 40, 45
- predict.rda, 50
- predict.sknn, 41, 55
- predict.svmlight, 42, 59
- print.greedy.wilks (*greedy.wilks*), 19
- print.loclda (*loclda*), 22
- print.meclight (*meclight.default*), 26
- print.pvs (*pvs*), 43
- print.rda (*rda*), 48
- print.stepclass (*stepclass*), 55
- pvs, 25, 26, 37, 40, 41, 43
- qbeta, 2
- qda, 12, 18, 29, 31, 39, 44, 50
- quadplot, 7, 46, 64
- quadtrafo, 47
- randomForest, 44
- rda, 12, 18, 19, 30, 31, 44, 48
- rpart, 12, 31
- sammon, 15
- scale, 4, 30, 55
- scatterplot3d, 46, 47
- shardsplot, 15, 51
- sknn, 12, 30, 31, 41, 42, 44, 54
- som, 51, 53
- step, 57
- stepAIC, 57
- stepclass, 20, 25, 43, 44, 55
- svm, 43, 44, 59
- svmlight, 12, 31, 42, 43, 57

table, 17
TopoS, 15, 53
triframe, 60, 61, 64
trigrd, 60, 60, 64, 66
trilines, 60–66
trilines (*tripoints*), 64
triperplines, 62, 64
tripplot, 7, 47, 60–62, 63, 65, 66
tripoints, 62, 64, 64, 66
tritrafo, 62, 64, 65, 65
ucpm, 55–57, 67