Package ‘e1071’

July 28, 2018

Version 1.7-0

Title Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien

Imports graphics, grDevices, class, stats, methods, utils

Suggests cluster, mlbench, nnet, randomForest, rpart, SparseM, xtable, Matrix, MASS

Description Functions for latent class analysis, short time Fourier transform, fuzzy clustering, support vector machines, shortest path computation, bagged clustering, naive Bayes classifier, ...

License GPL-2

LazyLoad yes

NeedsCompilation yes

Author David Meyer [aut, cre],
Evgenia Dimitriadou [aut, cph],
Kurt Hornik [aut],
Andreas Weingessel [aut],
Friedrich Leisch [aut],
Chih-Chung Chang [ctb, cph] (libsvm C++-code),
Chih-Chen Lin [ctb, cph] (libsvm C++-code)

Maintainer David Meyer <David.Meyer@R-project.org>

Repository CRAN

Date/Publication 2018-07-28 15:50:03 UTC

R topics documented:

`allShortestPaths` *Find Shortest Paths Between All Nodes in a Directed Graph*

Description

`allShortestPaths` finds all shortest paths in a directed (or undirected) graph using Floyd's algorithm. `extractPath` can be used to actually extract the path between a given pair of nodes.

Usage

```
allShortestPaths(x)
extractPath(obj, start, end)
```

Arguments

`x`	matrix or distance object
`obj`	return value of `allShortestPaths`
`start`	integer, starting point of path
`end`	integer, end point of path

Details

If `x` is a matrix, then `x[i,j]` has to be the length of the direct path from point `i` to point `j`. If no direct connection from point `i` to point `j` exist, then `x[i,j]` should be either `NA` or `Inf`. Note that the graph can be directed, hence `x[i,j]` need not be the same as `x[j,i]`. The main diagonal of `x` is ignored. Alternatively, `x` can be a distance object as returned by `dist` (corresponding to an undirected graph).

Value

`allShortestPaths` returns a list with components

`length`	A matrix with the total lengths of the shortest path between each pair of points.
`middlePoints`	A matrix giving a point in the middle of each shortest path (or 0 if the direct connection is the shortest path), this is mainly used as input for `extractPath`.

`extractPath` returns a vector of node numbers giving with the shortest path between two points.

Author(s)

Friedrich Leisch

References

Kumar, V., Grama, A., Gupta, A. and Karypis, G. Introduction to Parallel Programming - Design and Analysis of Algorithms, Benjamin Cummings Publishing, 1994, ISBN 0-8053-3170-0

Examples

```
## build a graph with 5 nodes
x <- matrix(NA, 5, 5)
diag(x) <- 0
x[1,2] <- 30; x[1,3] <- 10
x[2,4] <- 70; x[2,5] <- 40
x[3,4] <- 50; x[3,5] <- 20
x[4,5] <- 60
x[5,4] <- 10
print(x)

## compute all path lengths
z <- allShortestPaths(x)
print(z)

## the following should give 1 -> 3 -> 5 -> 4
extractPath(z, 1, 4)
```

---

bclust *Bagged Clustering*

Description

Cluster the data in x using the bagged clustering algorithm. A partitioning cluster algorithm such as kmeans is run repeatedly on bootstrap samples from the original data. The resulting cluster centers are then combined using the hierarchical cluster algorithm hclust.

Usage

```
bclust(x, centers=2, iter.base=10, minsize=0,
       dist.method="euclidian",
       hclust.method="average", base.method="kmeans",
       base.centers=20, verbose=TRUE,
       final.kmeans=FALSE, docmdscale=FALSE,
       resample=TRUE, weights=NULL, maxcluster=base.centers, ...)
hclust.bclust(object, x, centers, dist.method=object$dist.method,
              hclust.method=object$hclust.method, final.kmeans=FALSE,
              docmdscale = FALSE, maxcluster=object$maxcluster)
## S3 method for class 'bclust'
plot(x, maxcluster=x$maxcluster, main, ...)
centers.bclust(object, k)
clusters.bclust(object, k, x=NULL)
```

Arguments

x	Matrix of inputs (or object of class "bclust" for plot).
centers, k	Number of clusters.

iter.base	Number of runs of the base cluster algorithm.
minsize	Minimum number of points in a base cluster.
dist.method	Distance method used for the hierarchical clustering, see dist for available distances.
hclust.method	Linkage method used for the hierarchical clustering, see hclust for available methods.
base.method	Partitioning cluster method used as base algorithm.
base.centers	Number of centers used in each repetition of the base method.
verbose	Output status messages.
final.kmeans	If TRUE, a final kmeans step is performed using the output of the bagged clustering as initialization.
docmdscale	Logical, if TRUE a cmdscale result is included in the return value.
resample	Logical, if TRUE the base method is run on bootstrap samples of x, else directly on x.
weights	Vector of length nrow(x), weights for the resampling. By default all observations have equal weight.
maxcluster	Maximum number of clusters memberships are to be computed for.
object	Object of class "bclust".
main	Main title of the plot.
...	Optional arguments top be passed to the base method in bclust, ignored in plot.

Details

First, iter.base bootstrap samples of the original data in x are created by drawing with replacement. The base cluster method is run on each of these samples with base.centers centers. The base.method must be the name of a partitioning cluster function returning a list with the same components as the return value of kmeans.

This results in a collection of iter.base * base.centers centers, which are subsequently clustered using the hierarchical method hclust. Base centers with less than minsize points in there respective partitions are removed before the hierarchical clustering.

The resulting dendrogram is then cut to produce centers clusters. Hence, the name of the argument centers is a little bit misleading as the resulting clusters need not be convex, e.g., when single linkage is used. The name was chosen for compatibility with standard partitioning cluster methods such as kmeans.

A new hierarchical clustering (e.g., using another hclust.method) re-using previous base runs can be performed by running hclust.bclust on the return value of bclust.

Value

bclust and hclust.bclust return objects of class "bclust" including the components

hclust	Return value of the hierarchical clustering of the collection of base centers (Object of class "hclust").

`cluster`	Vector with indices of the clusters the inputs are assigned to.
`centers`	Matrix of centers of the final clusters. Only useful, if the hierarchical clustering method produces convex clusters.
`allcenters`	Matrix of all `iter.base * base.centers` centers found in the base runs.

Author(s)

Friedrich Leisch

References

Friedrich Leisch. Bagged clustering. Working Paper 51, SFB "Adaptive Information Systems and Modeling in Economics and Management Science", August 1999. http://epub.wu.ac.at/1272/1/document.pdf

See Also

`hclust`, `kmeans`, `boxplot.bclust`

Examples

```
data(iris)
bc1 <- bclust(iris[,1:4], 3, base.centers=5)
plot(bc1)

table(clusters.bclust(bc1, 3))
centers.bclust(bc1, 3)
```

`bincombinations` *Binary Combinations*

Description

Returns a matrix containing the 2^p vectors of length `p`.

Usage

```
bincombinations(p)
```

Arguments

`p`	Length of binary vectors

Author(s)

Friedrich Leisch

Examples

```
bincombinations(2)
bincombinations(3)
```

`bootstrap.lca` *Bootstrap Samples of LCA Results*

Description

This function draws bootstrap samples from a given LCA model and refits a new LCA model for each sample. The quality of fit of these models is compared to the original model.

Usage

```
bootstrap.lca(l, nsamples=10, lcaiter=30, verbose=FALSE)
```

Arguments

`l`	An LCA model as created by `lca`
`nsamples`	Number of bootstrap samples
`lcaiter`	Number of LCA iterations
`verbose`	If `TRUE` some output is printed during the computations.

Details

From a given LCA model `l`, `nsamples` bootstrap samples are drawn. For each sample a new LCA model is fitted. The goodness of fit for each model is computed via Likelihood Ratio and Pearson's Chisquare. The values for the fitted models are compared with the values of the original model `l`. By this method it can be tested whether the data to which `l` was originally fitted come from an LCA model.

Value

An object of class `bootstrap.lca` is returned, containing

`logl, loglsat`	The LogLikelihood of the models and of the corresponding saturated models
`lratio`	Likelihood quotient of the models and the corresponding saturated models
`lratiomean, lratiosd`	Mean and Standard deviation of `lratio`
`lratioorg`	Likelihood quotient of the original model and the corresponding saturated model
`zratio`	Z-Statistics of `lratioorg`
`pvalzratio, pvalratio`	P-Values for `zratio`, computed via normal distribution and empirical distribution
`chisq`	Pearson's Chisq of the models

`chisqmean, chisqsd`
: Mean and Standard deviation of `chisq`

`chisqorg`
: Pearson's Chisq of the original model

`zchisq`
: Z-Statistics of `chisqorg`

`pvalzchisq, pvalchisq`
: P-Values for `zchisq`, computed via normal distribution and empirical distribution

`nsamples`
: Number of bootstrap samples

`lcaiter`
: Number of LCA Iterations

Author(s)

Andreas Weingessel

References

Anton K. Formann: "Die Latent-Class-Analysis", Beltz Verlag 1984

See Also

`lca`

Examples

```
## Generate a 4-dim. sample with 2 latent classes of 500 data points each.
## The probabilities for the 2 classes are given by type1 and type2.
type1 <- c(0.8,0.8,0.2,0.2)
type2 <- c(0.2,0.2,0.8,0.8)
x <- matrix(runif(4000),nr=1000)
x[1:500,] <- t(t(x[1:500,])<type1)*1
x[501:1000,] <- t(t(x[501:1000,])<type2)*1

l <- lca(x, 2, niter=5)
bl <- bootstrap.lca(l,nsamples=3,lcaiter=5)
bl
```

`boxplot.bclust` *Boxplot of Cluster Profiles*

Description

Makes boxplots of the results of a bagged clustering run.

Usage

```
## S3 method for class 'bclust'
boxplot(x, n=nrow(x$centers), bycluster=TRUE,
        main=deparse(substitute(x)), oneplot=TRUE,
        which=1:n, ...)
```

Arguments

`x`	Clustering result, object of class `"bclust"`.
`n`	Number of clusters to plot, by default the number of clusters used in the call of `bclust`.
`bycluster`	If `TRUE` (default), a boxplot for each cluster is plotted. If `FALSE`, a boxplot for each variable is plotted.
`main`	Main title of the plot, by default the name of the cluster object.
`oneplot`	If `TRUE`, all boxplots appear on one screen (using an appropriate rectangular layout).
`which`	Number of clusters which should be plotted, default is all clusters.
`...`	Additional arguments for `boxplot`.

Author(s)

Friedrich Leisch

Examples

```
data(iris)
bc1 <- bclust(iris[,1:4], 3, base.centers=5)
boxplot(bc1)
```

`classAgreement` *Coefficients Comparing Classification Agreement*

Description

`classAgreement()` computes several coefficients of agreement between the columns and rows of a 2-way contingency table.

Usage

```
classAgreement(tab, match.names=FALSE)
```

Arguments

`tab`	A 2-dimensional contingency table.
`match.names`	Flag whether row and columns should be matched by name.

Details

Suppose we want to compare two classifications summarized by the contingency table $T = [t_{ij}]$ where $i, j = 1, \ldots, K$ and t_{ij} denotes the number of data points which are in class i in the first partition and in class j in the second partition. If both classifications use the same labels, then obviously the two classification agree completely if only elements in the main diagonal of the table are non-zero. On the other hand, large off-diagonal elements correspond to smaller agreement between the two classifications. If `match.names` is `TRUE`, the class labels as given by the row and column names are matched, i.e. only columns and rows with the same dimnames are used for the computation.

If the two classification do not use the same set of labels, or if identical labels can have different meaning (e.g., two outcomes of cluster analysis on the same data set), then the situation is a little bit more complicated. Let A denote the number of all pairs of data points which are either put into the same cluster by both partitions or put into different clusters by both partitions. Conversely, let D denote the number of all pairs of data points that are put into one cluster in one partition, but into different clusters by the other partition. Hence, the partitions disagree for all pairs D and agree for all pairs A. We can measure the agreement by the Rand index $A/(A + D)$ which is invariant with respect to permutations of the columns or rows of T.

Both indices have to be corrected for agreement by chance if the sizes of the classes are not uniform.

Value

A list with components

`diag`	Percentage of data points in the main diagonal of `tab`.
`kappa`	`diag` corrected for agreement by chance.
`rand`	Rand index.
`crand`	Rand index corrected for agreement by chance.

Author(s)

Friedrich Leisch

References

J.~Cohen. A coefficient of agreement for nominal scales. Educational and Psychological Measurement, 20, 37–46, 1960.

Lawrence Hubert and Phipps Arabie. Comparing partitions. Journal of Classification, 2, 193–218, 1985.

See Also

`matchClasses`

Examples

```
## no class correlations: both kappa and crand almost zero
g1 <- sample(1:5, size=1000, replace=TRUE)
g2 <- sample(1:5, size=1000, replace=TRUE)
tab <- table(g1, g2)
classAgreement(tab)

## let pairs (g1=1,g2=1) and (g1=3,g2=3) agree better
k <- sample(1:1000, size=200)
g1[k] <- 1
g2[k] <- 1

k <- sample(1:1000, size=200)
g1[k] <- 3
g2[k] <- 3

tab <- table(g1, g2)
## both kappa and crand should be significantly larger than before
classAgreement(tab)
```

`cmeans` *Fuzzy C-Means Clustering*

Description

The fuzzy version of the known *k*means clustering algorithm as well as an on-line variant (Unsupervised Fuzzy Competitive learning).

Usage

```
cmeans(x, centers, iter.max = 100, verbose = FALSE,
       dist = "euclidean", method = "cmeans", m = 2,
       rate.par = NULL, weights = 1, control = list())
```

Arguments

`x`	The data matrix where columns correspond to variables and rows to observations.
`centers`	Number of clusters or initial values for cluster centers.
`iter.max`	Maximum number of iterations.
`verbose`	If `TRUE`, make some output during learning.
`dist`	Must be one of the following: If `"euclidean"`, the mean square error, if `"manhattan"`, the mean absolute error is computed. Abbreviations are also accepted.
`method`	If `"cmeans"`, then we have the *c*-means fuzzy clustering method, if `"ufcl"` we have the on-line update. Abbreviations are also accepted.
`m`	A number greater than 1 giving the degree of fuzzification.

`rate.par`	A number between 0 and 1 giving the parameter of the learning rate for the on-line variant. The default corresponds to 0.3.
`weights`	a numeric vector with non-negative case weights. Recycled to the number of observations in `x` if necessary.
`control`	a list of control parameters. See **Details**.

Details

The data given by `x` is clustered by generalized versions of the fuzzy c-means algorithm, which use either a fixed-point or an on-line heuristic for minimizing the objective function

$$\sum_i \sum_j w_i u_{ij}^m d_{ij},$$

where w_i is the weight of observation i, u_{ij} is the membership of observation i in cluster j, and d_{ij} is the distance (dissimilarity) between observation i and center j. The dissimilarities used are the sums of squares (`"euclidean"`) or absolute values (`"manhattan"`) of the element-wise differences.

If `centers` is a matrix, its rows are taken as the initial cluster centers. If `centers` is an integer, `centers` rows of `x` are randomly chosen as initial values.

The algorithm stops when the maximum number of iterations (given by `iter.max`) is reached, or when the algorithm is unable to reduce the current value `val` of the objective function by `reltol * (abs(val) * reltol)` at a step. The relative convergence tolerance `reltol` can be specified as the `reltol` component of the list of control parameters, and defaults to `sqrt(.Machine$double.eps)`.

If `verbose` is `TRUE`, each iteration displays its number and the value of the objective function.

If `method` is `"cmeans"`, then we have the c-means fuzzy clustering method, see for example Bezdek (1981). If `"ufcl"`, we have the On-line Update (Unsupervised Fuzzy Competitive Learning) method due to Chung and Lee (1992), see also Pal et al (1996). This method works by performing an update directly after each input signal (i.e., for each single observation).

The parameters `m` defines the degree of fuzzification. It is defined for real values greater than 1 and the bigger it is the more fuzzy the membership values of the clustered data points are.

Value

An object of class `"fclust"` which is a list with components:

`centers`	the final cluster centers.
`size`	the number of data points in each cluster of the closest hard clustering.
`cluster`	a vector of integers containing the indices of the clusters where the data points are assigned to for the closest hard clustering, as obtained by assigning points to the (first) class with maximal membership.
`iter`	the number of iterations performed.
`membership`	a matrix with the membership values of the data points to the clusters.
`withinerror`	the value of the objective function.
`call`	the call used to create the object.

Author(s)

Evgenia Dimitriadou and Kurt Hornik

References

J. C. Bezdek (1981). *Pattern recognition with fuzzy objective function algorithms*. New York: Plenum.

Fu Lai Chung and Tong Lee (1992). Fuzzy competitive learning. *Neural Networks*, **7**(3), 539–551.

Nikhil R. Pal, James C. Bezdek, and Richard J. Hathaway (1996). Sequential competitive learning and the fuzzy c-means clustering algorithms. *Neural Networks*, **9**(5), 787–796.

Examples

```
# a 2-dimensional example
x<-rbind(matrix(rnorm(100,sd=0.3),ncol=2),
         matrix(rnorm(100,mean=1,sd=0.3),ncol=2))
cl<-cmeans(x,2,20,verbose=TRUE,method="cmeans",m=2)
print(cl)

# a 3-dimensional example
x<-rbind(matrix(rnorm(150,sd=0.3),ncol=3),
         matrix(rnorm(150,mean=1,sd=0.3),ncol=3),
         matrix(rnorm(150,mean=2,sd=0.3),ncol=3))
cl<-cmeans(x,6,20,verbose=TRUE,method="cmeans")
print(cl)
```

---

countpattern *Count Binary Patterns*

Description

Every row of the binary matrix `x` is transformed into a binary pattern and these patterns are counted.

Usage

```
countpattern(x, matching=FALSE)
```

Arguments

`x`	A matrix of binary observations
`matching`	If TRUE an additional vector is returned which stores which row belongs to which pattern

Value

A vector of length `2^ncol(x)` giving the number of times each pattern occurs in the rows of `x`. The names of this vector are the binary patterns. They are sorted according to their numeric value. If `matching` is `TRUE`, a list of the following two vectors is returned.

`pat`	Numbers of patterns as described above.
`matching`	Vector giving the position of the pattern of each row of `x` in `pat`.

Author(s)

Andreas Weingessel

Examples

```
xx <- rbind(c(1,0,0),c(1,0,0),c(1,0,1),c(0,1,1),c(0,1,1))
countpattern(xx)
countpattern(xx, matching=TRUE)
```

`cshell` *Fuzzy C-Shell Clustering*

Description

The *c*-shell clustering algorithm, the shell prototype-based version (ring prototypes) of the fuzzy *k*means clustering method.

Usage

```
cshell(x, centers, iter.max=100, verbose=FALSE, dist="euclidean",
       method="cshell", m=2, radius = NULL)
```

Arguments

`x`	The data matrix, were columns correspond to the variables and rows to observations.
`centers`	Number of clusters or initial values for cluster centers
`iter.max`	Maximum number of iterations
`verbose`	If `TRUE`, make some output during learning
`dist`	Must be one of the following: If `"euclidean"`, the mean square error, if `"manhattan"`, the mean absolute error is computed. Abbreviations are also accepted.
`method`	Currently, only the `"cshell"` method; the c-shell fuzzy clustering method
`m`	The degree of fuzzification. It is defined for values greater than *1*
`radius`	The radius of resulting clusters

Details

The data given by x is clustered by the fuzzy *c*-shell algorithm.

If centers is a matrix, its rows are taken as the initial cluster centers. If centers is an integer, centers rows of x are randomly chosen as initial values.

The algorithm stops when the maximum number of iterations (given by iter.max) is reached.

If verbose is TRUE, it displays for each iteration the number the value of the objective function.

If dist is "euclidean", the distance between the cluster center and the data points is the Euclidean distance (ordinary kmeans algorithm). If "manhattan", the distance between the cluster center and the data points is the sum of the absolute values of the distances of the coordinates.

If method is "cshell", then we have the *c*-shell fuzzy clustering method.

The parameters m defines the degree of fuzzification. It is defined for real values greater than 1 and the bigger it is the more fuzzy the membership values of the clustered data points are.

The parameter radius is by default set to *0.2* for every cluster.

Value

cshell returns an object of class "cshell".

centers	The final cluster centers.
size	The number of data points in each cluster.
cluster	Vector containing the indices of the clusters where the data points are assigned to. The maximum membership value of a point is considered for partitioning it to a cluster.
iter	The number of iterations performed.
membership	a matrix with the membership values of the data points to the clusters.
withinerror	Returns the sum of square distances within the clusters.
call	Returns a call in which all of the arguments are specified by their names.

Author(s)

Evgenia Dimitriadou

References

Rajesh N. Dave. *Fuzzy Shell-Clustering and Applications to Circle Detection in Digital Images.* Int. J. of General Systems, Vol. **16**, pp. 343-355, 1996.

Examples

```
## a 2-dimensional example
x<-rbind(matrix(rnorm(50,sd=0.3),ncol=2),
         matrix(rnorm(50,mean=1,sd=0.3),ncol=2))
cl<-cshell(x,2,20,verbose=TRUE,method="cshell",m=2)
print(cl)

# assign classes to some new data
```

```
y<-rbind(matrix(rnorm(13,sd=0.3),ncol=2),
         matrix(rnorm(13,mean=1,sd=0.3),ncol=2))
#         ycl<-predict(cl, y, type="both")
```

Discrete *Discrete Distribution*

Description

These functions provide information about the discrete distribution where the probability of the elements of `values` is proportional to the values given in `probs`, which are normalized to sum up to 1. `ddiscrete` gives the density, `pdiscrete` gives the distribution function, `qdiscrete` gives the quantile function and `rdiscrete` generates random deviates.

Usage

```
ddiscrete(x, probs, values = 1:length(probs))
pdiscrete(q, probs, values = 1:length(probs))
qdiscrete(p, probs, values = 1:length(probs))
rdiscrete(n, probs, values = 1:length(probs), ...)
```

Arguments

`x,q`	vector or array of quantiles.
`p`	vector or array of probabilities.
`n`	number of observations.
`probs`	probabilities of the distribution.
`values`	values of the distribution.
`...`	ignored (only there for backwards compatibility)

Details

The random number generator is simply a wrapper for `sample` and provided for backwards compatibility only.

Author(s)

Andreas Weingessel and Friedrich Leisch

Examples

```
## a vector of length 30 whose elements are 1 with probability 0.2
## and 2 with probability 0.8.
rdiscrete (30, c(0.2, 0.8))

## a vector of length 100 whose elements are A, B, C, D.
## The probabilities of the four values have the relation 1:2:3:3
rdiscrete (100, c(1,2,3,3), c("A","B","C","D"))
```

`e1071-deprecated` *Deprecated Functions in Package e1071*

Description

These functions are provided for compatibility with older versions of package **e1071** only, and may be defunct as soon as of the next release.

See Also

`Deprecated`

`element` *Extract Elements of an Array*

Description

Returns the element of `x` specified by `i`.

Usage

```
element(x, i)
```

Arguments

`x`	Array of arbitrary dimensionality.
`i`	Vector of the same length as `x` has dimension.

Author(s)

Friedrich Leisch

See Also

Extract

Examples

```
x <- array(1:20, dim=c(2,5,2))
element(x, c(1,4,2))
```

---

fclustIndex *Fuzzy Cluster Indexes (Validity/Performance Measures)*

Description

Calculates the values of several fuzzy validity measures. The values of the indexes can be independently used in order to evaluate and compare clustering partitions or even to determine the number of clusters existing in a data set.

Usage

```
fclustIndex(y, x, index = "all")
```

Arguments

y	An object of a fuzzy clustering result of class `"fclust"`
x	Data matrix
index	The validity measures used: `"gath.geva"`, `"xie.beni"`, `"fukuyama.sugeno"`, `"partition.coefficient"`, `"partition.entropy"`, `"proportion.exponent"`, `"separation.index"` and `"all"` for all the indexes.

Details

The validity measures and a short description of them follows, where N is the number of data points, u_{ij} the values of the membership matrix, v_j the centers of the clusters and k te number of clusters.

gath.geva: Gath and Geva introduced 2 main criteria for comparing and finding optimal partitions based on the heuristics that a better clustering assumes clear separation between the clusters, minimal volume of the clusters and maximal number of data points concentrated in the vicinity of the cluster centroids. These indexes are only for the cmeans clustering algorithm valid. For the first, the "fuzzy hypervolume" we have: $F_{HV} = \sum_{j=1}^{c} [\det(F_j)]^{1/2}$, where $F_j = \frac{\sum_{i=1}^{N} u_{ij}(x_i - v_j)(x_i - v_j)^T}{\sum_{i=1}^{N} u_{ij}}$, for the case when the defuzzification parameter is 2. For the second, the "average partition density": $D_{PA} = \frac{1}{k} \sum_{j=1}^{k} \frac{S_j}{[\det(F_j)]^{1/2}}$, where $S_j = \sum_{i=1}^{N} u_{ij}$. Moreover, the "partition density" which expresses the general partition density according to the physical definition of density is calculated by: $P_D = \frac{S}{F_{HV}}$, where $S = \sum_{j=1}^{k} \sum_{i=1}^{N} u_{ij}$.

xie.beni: This index is a function of the data set and the centroids of the clusters. Xie and Beni explained this index by writing it as a ratio of the total variation of the partition and the centroids (U,V) and the separation of the centroids vectors. The minimum values of this index under comparison support the best partitions. $u_{XB}(U, V; X) = \frac{\sum_{j=1}^{k} \sum_{i=1}^{N} u_{ij}^2 ||x_i - v_j||^2}{N(\min_{j \neq l}\{||v_j - v_l||^2\})}$

fukuyama.sugeno: This index consists of the difference of two terms, the first combining the fuzziness in the membership matrix with the geometrical compactness of the representation of the data set via the prototypes, and the second the fuzziness in its row of the partition matrix with the distance from the ith prototype to the grand mean of the data. The minimum values of this index also propose a good partition. $u_{FS}(U, V; X) = \sum_{i=1}^{N} \sum_{j=1}^{k} (u_{ij}^2)^q (||x_i - v_j||^2 - ||v_j - \bar{v}||^2)$

partition.coefficient: An index which measures the fuzziness of the partition but without considering the data set itself. It is a heuristic measure since it has no connection to any property of the data. The maximum values of it imply a good partition in the meaning of a least fuzzy clustering. $F(U; k) = \frac{tr(UU^T)}{N} = \frac{<U,U>}{N} = \frac{||U||^2}{N}$

- $F(U; k)$ shows the fuzziness or the overlap of the partition and depends on kN elements.
- $1/k \leq F(U; k) \leq 1$, where if $F(U; k) = 1$ then U is a hard partition and if $F(U; k) = 1/k$ then $U = [1/k]$ is the centroid of the fuzzy partion space P_{fk}. The converse is also valid.

partition.entropy: It is a measure that provides information about the membership matrix without also considering the data itself. The minimum values imply a good partition in the meaning of a more crisp partition. $H(U; k) = \sum_{i=1}^{N} h(u_i)/N$, where $h(u) = -\sum_{j=1}^{k} u_j \ \log_a(u_j)$ the Shannon's entropy.

- $H(U; k)$ shows the uncertainty of a fuzzy partition and depends also on kN elements. Specifically, $h(u_i)$ is interpreted as the amount of fuzzy information about the membership of x_i in k classes that is retained by column u_j. Thus, at $U = [1/k]$ the most information is withheld since the membership is the fuzziest possible.
- $0 \leq H(U; k) \leq \log_a(k)$, where for $H(U; k) = 0$ U is a hard partition and for $H(U; k) = \log_a(k)$ $U = [1/k]$.

proportion.exponent: It is a measure $P(U; k)$ of fuzziness adept to detect structural variations in the partition matrix as it becomes more fuzzier. A crisp cluster in the partition matrix can drive it to infinity when the partition coefficient and the partition entropy are more sensitive to small changes when approaching a hard partition. Its evaluation does not also involve the data or the algorithm used to partition them and its maximum implies the optimal partition but without knowing what maximum is a statistically significant maximum.

- $0 \leq P(U; k) < \infty$, since the $[0, 1]$ values explode to $[0, \infty)$ due to the natural logarithm. Specifically, $P = 0$ when and only when $U = [1/k]$, while $P \to \infty$ when any column of U is crisp.
- $P(U; k)$ can easily explode and it is good for partitions with large column maximums and at detecting structural variations.

separation.index (known as CS Index): This index identifies unique cluster structure with well-defined properties that depend on the data and a measure of distance. It answers the question if the clusters are compact and separated, but it rather seems computationally infeasible for big data sets since a distance matrix between all the data membership values has to be calculated. It also presupposes that a hard partition is derived from the fuzzy one.
$D_1(U; k; X, d) = \min_{i+1 \leq l \leq k-1} \left\{ \min_{1 \leq j \leq k} \left\{ \frac{dis(u_j, u_l)}{\max_{1 \leq m \leq k}\{dia(u_m)\}} \right\} \right\}$, where dia is the diameter of the subset, dis the distance of two subsets, and d a metric. U is a CS partition of $X \Leftrightarrow D_1 > 1$. When this holds then U is unique.

Value

Returns a vector with the validity measures values.

Author(s)

Evgenia Dimitriadou

References

James C. Bezdek, *Pattern Recognition with Fuzzy Objective Function Algorithms*, Plenum Press, 1981, NY.
L. X. Xie and G. Beni, *Validity measure for fuzzy clustering*, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. **3**, n. 8, p. 841-847, 1991.
I. Gath and A. B. Geva, *Unsupervised Optimal Fuzzy Clustering*, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. **11**, n. 7, p. 773-781, 1989.
Y. Fukuyama and M. Sugeno, *A new method of choosing the number of clusters for the fuzzy c-means method*, Proc. 5th Fuzzy Syst. Symp., p. 247-250, 1989 (in japanese).

See Also

cmeans

Examples

```
# a 2-dimensional example
x<-rbind(matrix(rnorm(100,sd=0.3),ncol=2),
         matrix(rnorm(100,mean=1,sd=0.3),ncol=2))
cl<-cmeans(x,2,20,verbose=TRUE,method="cmeans")
resultindexes <- fclustIndex(cl,x, index="all")
resultindexes
```

---

hamming.distance   *Hamming Distances of Vectors*

Description

If both x and y are vectors, hamming.distance returns the Hamming distance (number of different elements) between this two vectors. If x is a matrix, the Hamming distances between the rows of x are computed and y is ignored.

Usage

```
hamming.distance(x, y)
```

Arguments

x	a vector or matrix.
y	an optional vector.

Examples

```
x <- c(1, 0, 0)
y <- c(1, 0, 1)
hamming.distance(x, y)
z <- rbind(x,y)
rownames(z) <- c("Fred", "Tom")
hamming.distance(z)

hamming.distance(1:3, 3:1)
```

`hamming.window` *Computes the Coefficients of a Hamming Window.*

Description

The filter coefficients w_i of a Hamming window of length `n` are computed according to the formula

$$w_i = 0.54 - 0.46 \cos \frac{2\pi i}{n-1}$$

Usage

```
hamming.window(n)
```

Arguments

`n`	The length of the window.

Value

A vector containing the filter coefficients.

Author(s)

Andreas Weingessel

References

For a definition of the Hamming window, see for example
Alan V. Oppenheim and Roland W. Schafer: "Discrete-Time Signal Processing", Prentice-Hall, 1989.

See Also

stft, hanning.window

Examples

```
hamming.window(10)

x<-rnorm(500)
y<-stft(x, wtype="hamming.window")
plot(y)
```

`hanning.window` *Computes the Coefficients of a Hanning Window.*

Description

The filter coefficients w_i of a Hanning window of length `n` are computed according to the formula

$$w_i = 0.5 - 0.5 \cos \frac{2\pi i}{n-1}$$

Usage

```
hanning.window(n)
```

Arguments

`n`	The length of the window.

Value

A vector containing the filter coefficients.

Author(s)

Andreas Weingessel

References

For a definition of the Hanning window, see for example
Alan V. Oppenheim and Roland W. Schafer: "Discrete-Time Signal Processing", Prentice-Hall, 1989.

See Also

stft, hamming.window

Examples

```
hanning.window(10)

x<-rnorm(500)
y<-stft(x, wtype="hanning.window")
plot(y)
```

hsv_palette *Sequential color palette based on HSV colors*

Description

Computes a sequential color palette based on HSV colors by varying the saturation, given hue and value.

Usage

```
hsv_palette(h = 2/3, from = 0.7, to = 0.2, v = 1)
```

Arguments

h	hue
from	lower bound for saturation
to	upper bound for saturation
v	value

Value

A function with one argument: the size of the palette, i.e., the number of colors.

Author(s)

David Meyer <David.Meyer@R-project.org>

See Also

hsv

Examples

```
pie(rep(1, 10), col = hsv_palette()(10))
pie(rep(1, 10), col = hsv_palette(h = 0)(10))
```

ica *Independent Component Analysis*

Description

This is an R-implementation of the Matlab-Function of Petteri.Pajunen@hut.fi.

For a data matrix X independent components are extracted by applying a nonlinear PCA algorithm. The parameter `fun` determines which nonlinearity is used. `fun` can either be a function or one of the following strings "negative kurtosis", "positive kurtosis", "4th moment" which can be abbreviated to uniqueness. If `fun` equals "negative (positive) kurtosis" the function tanh (x-tanh(x)) is used which provides ICA for sources with negative (positive) kurtosis. For `fun == "4th moments"` the signed square function is used.

Usage

```
ica(X, lrate, epochs=100, ncomp=dim(X)[2], fun="negative")
```

Arguments

`X`	The matrix for which the ICA is to be computed
`lrate`	learning rate
`epochs`	number of iterations
`ncomp`	number of independent components
`fun`	function used for the nonlinear computation part

Value

An object of class `"ica"` which is a list with components

`weights`	ICA weight matrix
`projection`	Projected data
`epochs`	Number of iterations
`fun`	Name of the used function
`lrate`	Learning rate used
`initweights`	Initial weight matrix

Note

Currently, there is no reconstruction from the ICA subspace to the original input space.

Author(s)

Andreas Weingessel

References

Oja et al., “Learning in Nonlinear Constrained Hebbian Networks”, in Proc. ICANN-91, pp. 385–390.

Karhunen and Joutsensalo, “Generalizations of Principal Component Analysis, Optimization Problems, and Neural Networks”, Neural Networks, v. 8, no. 4, pp. 549–562, 1995.

`impute` *Replace Missing Values*

Description

Replaces missing values of a matrix or dataframe with the medians (`what="median"`) or means (`what="mean"`) of the respective columns.

Usage

```
impute(x, what = c("median", "mean"))
```

Arguments

`x`	A matrix or dataframe.
`what`	What to impute.

Value

A matrix or dataframe.

Author(s)

Friedrich Leisch

Examples

```
x<- matrix(1:10, ncol=2)
x[c(1,3,7)] <- NA
print(x)
print(impute(x))
```

interpolate *Interpolate Values of Array*

Description

For each row in matrix x, the hypercube of a containing this point is searched. The corners of the hypercube are linearly interpolated. By default, dimnames(a) is taken to contain the coordinate values for each point in a. This can be overridden using adims. If method=="constant", the value of the "lower left" corner of the hypercube is returned.

Usage

```
interpolate(x, a, adims=lapply(dimnames(a), as.numeric),
            method="linear")
```

Arguments

x	Matrix of values at which interpolation shall take place.
a	Array of arbitrary dimension.
adims	List of the same structure as dimnames(a).
method	Interpolation method, one of "linear" or "constant".

Author(s)

Friedrich Leisch

See Also

approx, spline

Examples

```
x <- seq(0,3,0.2)
z <- outer(x,x, function(x,y) sin(x*y))
dimnames(z) <- list(x,x)
sin(1.1*2.1)
interpolate(c(1.1, 2.1),z)
```

`kurtosis` *Kurtosis*

Description

Computes the kurtosis.

Usage

```
kurtosis(x, na.rm = FALSE, type = 3)
```

Arguments

`x`	a numeric vector containing the values whose kurtosis is to be computed.
`na.rm`	a logical value indicating whether `NA` values should be stripped before the computation proceeds.
`type`	an integer between 1 and 3 selecting one of the algorithms for computing skewness detailed below.

Details

If `x` contains missings and these are not removed, the skewness is `NA`.

Otherwise, write x_i for the non-missing elements of `x`, n for their number, μ for their mean, s for their standard deviation, and $m_r = \sum_i (x_i - \mu)^r / n$ for the sample moments of order r.

Joanes and Gill (1998) discuss three methods for estimating kurtosis:

Type 1: $g_2 = m_4/m_2^2 - 3$. This is the typical definition used in many older textbooks.

Type 2: $G_2 = ((n+1)g_2 + 6) * (n-1)/((n-2)(n-3))$. Used in SAS and SPSS.

Type 3: $b_2 = m_4/s^4 - 3 = (g_2 + 3)(1 - 1/n)^2 - 3$. Used in MINITAB and BMDP.

Only G_2 (corresponding to `type = 2`) is unbiased under normality.

Value

The estimated kurtosis of `x`.

References

D. N. Joanes and C. A. Gill (1998), Comparing measures of sample skewness and kurtosis. *The Statistician*, **47**, 183–189.

Examples

```
x <- rnorm(100)
kurtosis(x)
```

`lca` *Latent Class Analysis (LCA)*

Description

A latent class analysis with `k` classes is performed on the data given by `x`.

Usage

```
lca(x, k, niter=100, matchdata=FALSE, verbose=FALSE)
```

Arguments

`x`	Either a data matrix of binary observations or a list of patterns as created by `countpattern`
`k`	Number of classes used for LCA
`niter`	Number of Iterations
`matchdata`	If `TRUE` and `x` is a data matrix, the class membership of every data point is returned, otherwise the class membership of every pattern is returned.
`verbose`	If `TRUE` some output is printed during the computations.

Value

An object of class `"lca"` is returned, containing

`w`	Probabilities to belong to each class
`p`	Probabilities of a '1' for each variable in each class
`matching`	Depending on `matchdata` either the class membership of each pattern or of each data point
`logl, loglsat`	The LogLikelihood of the model and of the saturated model
`bic, bicsat`	The BIC of the model and of the saturated model
`chisq`	Pearson's Chisq
`lhquot`	Likelihood quotient of the model and the saturated model
`n`	Number of data points.
`np`	Number of free parameters.

Author(s)

Andreas Weingessel

References

Anton K. Formann: "Die Latent-Class-Analysis", Beltz Verlag 1984

See Also

countpattern, bootstrap.lca

Examples

```
## Generate a 4-dim. sample with 2 latent classes of 500 data points each.
## The probabilities for the 2 classes are given by type1 and type2.
type1 <- c(0.8,0.8,0.2,0.2)
type2 <- c(0.2,0.2,0.8,0.8)
x <- matrix(runif(4000),nr=1000)
x[1:500,] <- t(t(x[1:500,])<type1)*1
x[501:1000,] <- t(t(x[501:1000,])<type2)*1

l <- lca(x, 2, niter=5)
print(l)
summary(l)
p <- predict(l, x)
table(p, c(rep(1,500),rep(2,500)))
```

---

matchClasses *Find Similar Classes in Two-way Contingency Tables*

Description

Try to find a mapping between the two groupings, such that as many cases as possible are in one of the matched pairs.

Usage

```
matchClasses(tab, method="rowmax", iter=1, maxexact=9, verbose=TRUE)
compareMatchedClasses(x, y, method="rowmax", iter=1,
                      maxexact=9, verbose=FALSE)
```

Arguments

tab	Two-way contingency table of class memberships
method	One of "rowmax", "greedy" or "exact".
iter	Number of iterations used in greedy search.
verbose	If TRUE, display some status messages during computation.
maxexact	Maximum number of variables for which all possible permutations are computed.
x, y	Vectors or matrices with class memberships.

Details

If `method="rowmax"`, then each class defining a row in the contingency table is mapped to the column of the corresponding row maximum. Hence, some columns may be mapped to more than one row (while each row is mapped to a single column).

If `method="greedy"` or `method="exact"`, then the contingency table must be a square matrix and a unique mapping is computed. This corresponds to a permutation of columns and rows, such that sum of the main diagonal, i.e., the trace of the matrix, gets as large as possible. For both methods, first all pairs where row and columns maxima correspond and are bigger than the sum of all other elements in the corresponding columns and rows together are located and fixed (this is a necessary condition for maximal trace).

If `method="exact"`, then for the remaining rows and columns, all possible permutations are computed and the optimum is returned. This can get computationally infeasible very fast. If more than `maxexact` rows and columns remain after applying the necessary condition, then `method` is reset to `"greedy"`. If `method="greedy"`, then a greedy heuristic is tried `iter` times. Repeatedly a row is picked at random and matched to the free column with the maximum value.

`compareMatchedClasses()` computes the contingency table for each combination of columns from `x` and `y` and applies `matchClasses` to that table. The columns of the table are permuted accordingly and then the table is passed to `classAgreement`. The resulting agreement coefficients (diag, kappa, `...`) are returned. The return value of `compareMatchedClasses()` is a list containing a matrix for each coefficient; with element (k,l) corresponding to the k-th column of `x` and l-th column of `y`. If `y` is missing, then the columns of `x` are compared with each other.

Author(s)

Friedrich Leisch

See Also

`classAgreement`

Examples

```
## a stupid example with no class correlations:
g1 <- sample(1:5, size=1000, replace=TRUE)
g2 <- sample(1:5, size=1000, replace=TRUE)
tab <- table(g1, g2)
matchClasses(tab, "exact")

## let pairs (g1=1,g2=4) and (g1=3,g2=1) agree better
k <- sample(1:1000, size=200)
g1[k] <- 1
g2[k] <- 4

k <- sample(1:1000, size=200)
g1[k] <- 3
g2[k] <- 1

tab <- table(g1, g2)
matchClasses(tab, "exact")
```

```
## get agreement coefficients:
compareMatchedClasses(g1, g2, method="exact")
```

`matchControls` *Find Matched Control Group*

Description

Finds controls matching the cases as good as possible.

Usage

```
matchControls(formula, data = list(), subset, contlabel = "con",
              caselabel = NULL, dogrep = TRUE, replace = FALSE)
```

Arguments

`formula`	A formula indicating cases, controls and the variables to be matched. Details are described below.
`data`	an optional data frame containing the variables in the model. By default the variables are taken from the environment which `matchControls` is called from.
`subset`	an optional vector specifying a subset of observations to be used in the matching process.
`contlabel`	A string giving the label of the control group.
`caselabel`	A string giving the labels of the cases.
`dogrep`	If `TRUE`, then `contlabel` and `contlabel` are matched using `grep`, else string comparison (exact equality) is used.
`replace`	If `FALSE`, then every control is used only once.

Details

The left hand side of the `formula` must be a factor determining whether an observation belongs to the case or the control group. By default, all observations where a grep of `contlabel` matches, are used as possible controls, the rest is taken as cases. If `caselabel` is given, then only those observations are taken as cases. If `dogrep = TRUE`, then both `contlabel` and `caselabel` can be regular expressions.

The right hand side of the `formula` gives the variables that should be matched. The matching is done using the `daisy` distance from the `cluster` package, i.e., a model frame is built from the formula and used as input for `daisy`. For each case, the nearest control is selected. If `replace = FALSE`, each control is used only once.

Value

Returns a list with components

`cases`	Row names of cases.
`controls`	Row names of matched controls.
`factor`	A factor with 2 levels indicating cases and controls (the rest is set to `NA`.

Author(s)

Friedrich Leisch

Examples

```
Age.case <- 40 + 5 * rnorm(50)
Age.cont <- 45 + 10 * rnorm(150)
Age <- c(Age.case, Age.cont)

Sex.case <- sample(c("M", "F"), 50, prob = c(.4, .6), replace = TRUE)
Sex.cont <- sample(c("M", "F"), 150, prob = c(.6, .4), replace = TRUE)
Sex <- as.factor(c(Sex.case, Sex.cont))

casecont <- as.factor(c(rep("case", 50), rep("cont", 150)))

## now look at the group properties:
boxplot(Age ~ casecont)
barplot(table(Sex, casecont), beside = TRUE)

m <- matchControls(casecont ~ Sex + Age)

## properties of the new groups:
boxplot(Age ~ m$factor)
barplot(table(Sex, m$factor))
```

`moment` *Statistical Moment*

Description

Computes the (optionally centered and/or absolute) sample moment of a certain order.

Usage

```
moment(x, order=1, center=FALSE, absolute=FALSE, na.rm=FALSE)
```

Arguments

x	a numeric vector containing the values whose moment is to be computed.
order	order of the moment to be computed, the default is to compute the first moment, i.e., the mean.
center	a logical value indicating whether centered moments are to be computed.
absolute	a logical value indicating whether absolute moments are to be computed.
na.rm	a logical value indicating whether NA values should be stripped before the computation proceeds.

Details

When center and absolute are both FALSE, the moment is simply sum(x ^ order) / length(x).

Author(s)

Kurt Hornik and Friedrich Leisch

See Also

mean, var

Examples

```
x <- rnorm(100)

## Compute the mean
moment(x)
## Compute the 2nd centered moment (!= var)
moment(x, order=2, center=TRUE)

## Compute the 3rd absolute centered moment
moment(x, order=3, center=TRUE, absolute=TRUE)
```

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, laplace = 0, ..., subset, na.action = na.pass)
## Default S3 method:
naiveBayes(x, y, laplace = 0, ...)


## S3 method for class 'naiveBayes'
predict(object, newdata,
  type = c("class", "raw"), threshold = 0.001, eps = 0, ...)
```

Arguments

`x`	A numeric matrix, or a data frame of categorical and/or numeric variables.
`y`	Class vector.
`formula`	A formula of the form `class ~ x1 + x2 + ...`. Interactions are not allowed.
`data`	Either a data frame of predictors (categorical and/or numeric) or a contingency table.
`laplace`	positive double controlling Laplace smoothing. The default (0) disables Laplace smoothing.
`...`	Currently not used.
`subset`	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.)
`na.action`	A function to specify the action to be taken if `NA`s are found. The default action is not to count them for the computation of the probability factors. 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.)
`object`	An object of class `"naiveBayes"`.
`newdata`	A dataframe with new predictors (with possibly fewer columns than the training data). Note that the column names of `newdata` are matched against the training data ones.
`type`	If `"raw"`, the conditional a-posterior probabilities for each class are returned, and the class with maximal probability else.
`threshold`	Value replacing cells with probabilities within `eps` range.
`eps`	double for specifying an epsilon-range to apply laplace smoothing (to replace zero or close-zero probabilities by `theshold`.)

Details

The standard naive Bayes classifier (at least this implementation) assumes independence of the predictor variables, and Gaussian distribution (given the target class) of metric predictors. For attributes with missing values, the corresponding table entries are omitted for prediction.

Value

An object of class "naiveBayes" including components:

apriori
: Class distribution for the dependent variable.

tables
: 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.

Author(s)

David Meyer <David.Meyer@R-project.org>. Laplace smoothing enhancement by Jinghao Xue.

Examples

```
## Categorical data only:
data(HouseVotes84, package = "mlbench")
model <- naiveBayes(Class ~ ., data = HouseVotes84)
predict(model, HouseVotes84[1:10,])
predict(model, HouseVotes84[1:10,], type = "raw")

pred <- predict(model, HouseVotes84)
table(pred, HouseVotes84$Class)

## using laplace smoothing:
model <- naiveBayes(Class ~ ., data = HouseVotes84, laplace = 3)
pred <- predict(model, HouseVotes84[,-1])
table(pred, HouseVotes84$Class)


## Example of using a contingency table:
data(Titanic)
m <- naiveBayes(Survived ~ ., data = Titanic)
m
predict(m, as.data.frame(Titanic))

## Example with metric predictors:
data(iris)
m <- naiveBayes(Species ~ ., data = iris)
## alternatively:
m <- naiveBayes(iris[,-5], iris[,5])
m
table(predict(m, iris), iris[,5])
```

---

permutations — *All Permutations of Integers 1:n*

Description

Returns a matrix containing all permutations of the integers `1:n` (one permutation per row).

Usage

```
permutations(n)
```

Arguments

`n`	Number of element to permute.

Author(s)

Friedrich Leisch

Examples

```
permutations(3)
```

`plot.stft` *Plot Short Time Fourier Transforms*

Description

An object of class `"stft"` is plotted as a gray scale image. The x-axis corresponds to time, the y-axis to frequency. If the default colormap is used, dark regions in the plot correspond to high values at the particular time/frequency location.

Usage

```
## S3 method for class 'stft'
plot(x, col = gray(63:0/63), ...)
```

Arguments

`x`	An object of class `"stft"` as obtained by the function `stft`.
`col`	An optional colormap. By default 64 gray values are used, where white corresponds to the minimum value and black to the maximum.
`...`	further arguments to be passed to or from methods.

Value

No return value. This function is only for plotting.

Author(s)

Andreas Weingessel

See Also

stft

Examples

```
x<-rnorm(500)
y<-stft(x)
plot(y)
```

`plot.svm` *Plot SVM Objects*

Description

Generates a scatter plot of the input data of a `svm` fit for classification models by highlighting the classes and support vectors. Optionally, draws a filled contour plot of the class regions.

Usage

```
## S3 method for class 'svm'
plot(x, data, formula, fill = TRUE, grid = 50, slice = list(),
symbolPalette = palette(), svSymbol = "x", dataSymbol = "o", ...)
```

Arguments

`x`	An object of class `svm`
`data`	data to visualize. Should be the same used for fitting.
`formula`	formula selecting the visualized two dimensions. Only needed if more than two input variables are used.
`fill`	switch indicating whether a contour plot for the class regions should be added.
`grid`	granularity for the contour plot.
`slice`	a list of named values for the dimensions held constant (only needed if more than two variables are used). The defaults for unspecified dimensions are 0 (for numeric variables) and the first level (for factors). Factor levels can either be specified as factors or character vectors of length 1.
`symbolPalette`	Color palette used for the class the data points and support vectors belong to.
`svSymbol`	Symbol used for support vectors.
`dataSymbol`	Symbol used for data points (other than support vectors).
`...`	additional graphics parameters passed to `filled.contour` and `plot`.

Author(s)

David Meyer
`<David.Meyer@R-project.org>`

See Also

`svm`

Examples

```
## a simple example
data(cats, package = "MASS")
m <- svm(Sex~., data = cats)
plot(m, cats)

## more than two variables: fix 2 dimensions
data(iris)
m2 <- svm(Species~., data = iris)
plot(m2, iris, Petal.Width ~ Petal.Length,
     slice = list(Sepal.Width = 3, Sepal.Length = 4))

## plot with custom symbols and colors
plot(m, cats, svSymbol = 1, dataSymbol = 2, symbolPalette = rainbow(4),
color.palette = terrain.colors)
```

`plot.tune` *Plot Tuning Object*

Description

Visualizes the results of parameter tuning.

Usage

```
## S3 method for class 'tune'
plot(x, type = c("contour", "perspective"), theta = 60,
          col = "lightblue", main = NULL, xlab = NULL, ylab = NULL,
          swapxy = FALSE, transform.x = NULL, transform.y = NULL,
          transform.z = NULL, color.palette = hsv_palette(),
          nlevels = 20, ...)
```

Arguments

`x`	an object of class `tune`
`type`	choose whether a contour plot or a perspective plot is used if two parameters are to be visualized. Ignored if only one parameter has been tuned.
`theta`	angle of azimuthal direction.
`col`	the color(s) of the surface facets. Transparent colors are ignored.
`main`	main title
`xlab, ylab`	titles for the axes. N.B. These must be character strings; expressions are not accepted. Numbers will be coerced to character strings.
`swapxy`	if `TRUE`, the parameter axes are swaped (only used in case of two parameters).

`transform.x, transform.y, transform.z`
: functions to transform the parameters (x and y) and the error measures (z). Ignored if NULL.

`color.palette` color palette used in contour plot.

`nlevels` number of levels used in contour plot.

`...` Further graphics parameters.

Author(s)

David Meyer (based on C/C++-code by Chih-Chung Chang and Chih-Jen Lin)
<David.Meyer@R-project.org>

See Also

tune

Examples

```
data(iris)
obj <- tune.svm(Species~., data = iris, sampling = "fix",
                gamma = 2^c(-8,-4,0,4), cost = 2^c(-8,-4,-2,0))
plot(obj, transform.x = log2, transform.y = log2)
plot(obj, type = "perspective", theta = 120, phi = 45)
```

---

predict.svm *Predict Method for Support Vector Machines*

Description

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

Usage

```
## S3 method for class 'svm'
predict(object, newdata, decision.values = FALSE,
probability = FALSE, ..., na.action = na.omit)
```

Arguments

`object` Object of class "svm", created by svm.

`newdata` An object containing the new input data: either a matrix or a sparse matrix (object of class Matrix provided by the **Matrix** package, or of class matrix.csr provided by the **SparseM** package, or of class simple_triplet_matrix provided by the **slam** package). A vector will be transformed to a n x 1 matrix.

`decision.values`
: Logical controlling whether the decision values of all binary classifiers computed in multiclass classification shall be computed and returned.

`probability` Logical indicating whether class probabilities should be computed and returned. Only possible if the model was fitted with the `probability` option enabled.

`na.action` A function to specify the action to be taken if 'NA's are found. The default action is `na.omit`, which leads to rejection of cases with missing values on any required variable. An alternative is `na.fail`, which causes an error if `NA` cases are found. (NOTE: If given, this argument must be named.)

`...` Currently not used.

Value

A vector of predicted values (for classification: a vector of labels, for density estimation: a logical vector). If `decision.value` is `TRUE`, the vector gets a `"decision.values"` attribute containing a n x c matrix (n number of predicted values, c number of classifiers) of all c binary classifiers' decision values. There are k * (k - 1) / 2 classifiers (k number of classes). The colnames of the matrix indicate the labels of the two classes. If `probability` is `TRUE`, the vector gets a `"probabilities"` attribute containing a n x k matrix (n number of predicted values, k number of classes) of the class probabilities.

Note

If the training set was scaled by `svm` (done by default), the new data is scaled accordingly using scale and center of the training data.

Author(s)

David Meyer (based on C++-code by Chih-Chung Chang and Chih-Jen Lin)
`<David.Meyer@R-project.org>`

See Also

`svm`

Examples

```
data(iris)
attach(iris)

## classification mode
# default with factor response:
model <- svm(Species ~ ., data = iris)

# alternatively the traditional interface:
x <- subset(iris, select = -Species)
y <- Species
model <- svm(x, y, probability = TRUE)

print(model)
summary(model)

# test with train data
```

```
pred <- predict(model, x)
# (same as:)
pred <- fitted(model)

# compute decision values and probabilites
pred <- predict(model, x, decision.values = TRUE, probability = TRUE)
attr(pred, "decision.values")[1:4,]
attr(pred, "probabilities")[1:4,]

## try regression mode on two dimensions

# create data
x <- seq(0.1, 5, by = 0.05)
y <- log(x) + rnorm(x, sd = 0.2)

# estimate model and predict input values
m   <- svm(x, y)
new <- predict(m, x)

# visualize
plot   (x, y)
points (x, log(x), col = 2)
points (x, new, col = 4)

## density-estimation

# create 2-dim. normal with rho=0:
X <- data.frame(a = rnorm(1000), b = rnorm(1000))
attach(X)

# traditional way:
m <- svm(X, gamma = 0.1)

# formula interface:
m <- svm(~., data = X, gamma = 0.1)
# or:
m <- svm(~ a + b, gamma = 0.1)

# test:
newdata <- data.frame(a = c(0, 4), b = c(0, 4))
predict (m, newdata)

# visualize:
plot(X, col = 1:1000 %in% m$index + 1, xlim = c(-5,5), ylim=c(-5,5))
points(newdata, pch = "+", col = 2, cex = 5)
```

`probplot` *Probability Plot*

Description

Generates a probability plot for a specified theoretical distribution, i.e., basically a qqplot where the y-axis is labeled with probabilities instead of quantiles. The function is mainly intended for teaching the concept of quantile plots.

Usage

```
probplot(x, qdist=qnorm, probs=NULL, line=TRUE,
          xlab=NULL, ylab="Probability in %", ...)
## S3 method for class 'probplot'
lines(x, h=NULL, v=NULL, bend=FALSE, ...)
```

Arguments

`x`	A data vector for `probplot`, an object of class `probplot` for the `lines` method.
`qdist`	A character string or a function for the quantiles of the target distribution.
`probs`	Vector of probabilities at which horizontal lines should be drawn.
`line`	Add a line passing through the quartiles to the plot?
`xlab, ylab`	Graphical parameters.
`h`	The y-value for a horizontal line.
`v`	The x-value for a vertical line.
`bend`	If `TRUE`, lines are "bent" at the quartile line, else regular `ablines` are added. See examples.
`...`	Further arguments for `qdist` and graphical parameters for lines.

Author(s)

Friedrich Leisch

See Also

qqplot

Examples

```
## a simple example
x <- rnorm(100, mean=5)
probplot(x)

## the same with horizontal tickmarks at the y-axis
opar <- par("las")
par(las=1)
probplot(x)

## this should show the lack of fit at the tails
probplot(x, "qunif")

## for increasing degrees of freedom the t-distribution converges to
```

```
## normal
probplot(x, qt, df=1)
probplot(x, qt, df=3)
probplot(x, qt, df=10)
probplot(x, qt, df=100)

## manually add the line through the quartiles
p <- probplot(x, line=FALSE)
lines(p, col="green", lty=2, lwd=2)


## Make the line at prob=0.5 red
lines(p, h=0.5, col="red")

### The following use the estimted distribution given by the green
### line:

## What is the probability that x is smaller than 7?
lines(p, v=7, bend=TRUE, col="blue")

## Median and 90% confidence interval
lines(p, h=.5, col="red", lwd=3, bend=TRUE)
lines(p, h=c(.05, .95), col="red", lwd=2, lty=3, bend=TRUE)

par(opar)
```

---

rbridge *Simulation of Brownian Bridge*

Description

rwiener returns a time series containing a simulated realization of the Brownian bridge on the interval [0,end]. If W(t) is a Wiener process, then the Brownian bridge is defined as W(t) - t W(1).

Usage

```
rbridge(end = 1, frequency = 1000)
```

Arguments

end	the time of the last observation.
frequency	the number of observations per unit of time.

See Also

rwiener

Examples

```
# simulate a Brownian bridge on [0,1] and plot it

x <- rbridge()
plot(x,type="l")
```

---

read.matrix.csr *Read/Write Sparse Data*

Description

reads and writes a file in sparse data format.

Usage

```
read.matrix.csr(file, fac = TRUE, ncol = NULL)
write.matrix.csr(x, file = "out.dat", y = NULL, fac = TRUE)
```

Arguments

`x`	An object of class `matrix.csr`
`y`	A vector (either numeric or a factor)
`file`	The filename.
`fac`	If `TRUE`, the y-values (if any) are interpreted as factor levels.
`ncol`	Number of columns, detected automatically. Can be used to add empty columns (possibly not stored in the sparse format).

Value

If the data file includes no `y` variable, `read.matrix.csr` returns an object of class `matrix.csr`, else a list with components:

`x`	object of class `matrix.csr`
`y`	vector of numeric values or factor levels, depending on `fac`.

Author(s)

David Meyer
`<David.Meyer@R-project.org>`

See Also

`matrix.csr`

Examples

```
## Not run:
library(methods)
if (require(SparseM)) {
    data(iris)
    x <- as.matrix(iris[,1:4])
    y <- iris[,5]
    xs <- as.matrix.csr(x)
    write.matrix.csr(xs, y = y, file = "iris.dat")
    xs2 <- read.matrix.csr("iris.dat")$x
    if (!all(as.matrix(xs) == as.matrix(xs2)))
        stop("Error: objects are not equal!")
}

## End(Not run)
```

`rectangle.window` *Computes the Coefficients of a Rectangle Window.*

Description

Returns the filter coefficients of a rectangle window. That is a vector of `n` 1.

The purpose of this function is just to have a name for the R command `rep (1, n)`.

Usage

```
rectangle.window(n)
```

Arguments

`n`	The length of the window.

Value

A vector of length `n` filled with 1.

Author(s)

Andreas Weingessel

See Also

stft

Examples

```
x<-rnorm(500)
y<-stft(x, wtype="rectangle.window")
plot(y)
```

`rwiener` *Simulation of Wiener Process*

Description

`rwiener` returns a time series containing a simulated realization of the Wiener process on the interval [0,end]

Usage

```
rwiener(end = 1, frequency = 1000)
```

Arguments

`end`	the time of the last observation.
`frequency`	the number of observations per unit of time.

Examples

```
# simulate a Wiener process on [0,1] and plot it

x <- rwiener()
plot(x,type="l")
```

`sigmoid` *The Logistic Function and Derivatives*

Description

Sigmoid $1/(1 + \exp(-x))$, first and second derivative.

Usage

```
sigmoid(x)
dsigmoid(x)
d2sigmoid(x)
```

Arguments

`x`	a numeric vector

Author(s)

Friedrich Leisch

Examples

```
plot(sigmoid, -5, 5, ylim = c(-.2, 1))
plot(dsigmoid, -5, 5, add = TRUE, col = 2)
plot(d2sigmoid, -5, 5, add = TRUE, col = 3)
```

---

skewness *Skewness*

Description

Computes the skewness.

Usage

```
skewness(x, na.rm = FALSE, type = 3)
```

Arguments

`x`	a numeric vector containing the values whose skewness is to be computed.
`na.rm`	a logical value indicating whether `NA` values should be stripped before the computation proceeds.
`type`	an integer between 1 and 3 selecting one of the algorithms for computing skewness detailed below.

Details

If `x` contains missings and these are not removed, the skewness is `NA`.

Otherwise, write x_i for the non-missing elements of `x`, n for their number, μ for their mean, s for their standard deviation, and $m_r = \sum_i (x_i - \mu)^r / n$ for the sample moments of order r.

Joanes and Gill (1998) discuss three methods for estimating skewness:

Type 1: $g_1 = m_3 / m_2^{3/2}$. This is the typical definition used in many older textbooks.

Type 2: $G_1 = g_1 \sqrt{n(n-1)}/(n-2)$. Used in SAS and SPSS.

Type 3: $b_1 = m_3/s^3 = g_1((n-1)/n)^{3/2}$. Used in MINITAB and BMDP.

All three skewness measures are unbiased under normality.

Value

The estimated skewness of `x`.

References

D. N. Joanes and C. A. Gill (1998), Comparing measures of sample skewness and kurtosis. *The Statistician*, **47**, 183–189.

Examples

```
x <- rnorm(100)
skewness(x)
```

`stft` *Computes the Short Time Fourier Transform of a Vector*

Description

This function computes the Short Time Fourier Transform of a given vector `X`.

First, time-slices of length `win` are extracted from the vector. The shift of one time-slice to the next one is given by `inc`. The values of these time-slices are smoothed by mulitplying them with a window function specified in `wtype`. For the thus obtained windows, the Fast Fourier Transform is computed.

Usage

```
stft(X, win=min(80,floor(length(X)/10)), inc=min(24,
floor(length(X)/30)), coef=64, wtype="hanning.window")
```

Arguments

`X`	The vector from which the stft is computed.
`win`	Length of the window. For long vectors the default window size is 80, for short vectors the window size is chosen so that 10 windows fit in the vector.
`inc`	Increment by which the window is shifted. For long vectors the default increment is 24, for short vectors the increment is chosen so that 30 increments fit in the vector.
`coef`	Number of Fourier coefficients
`wtype`	Type of window used

Value

Object of type stft. Contains the values of the stft and information about the parameters.

`values`	A matrix containing the results of the stft. Each row of the matrix contains the `coef` Fourier coefficients of one window.
`windowsize`	The value of the parameter `win`
`increment`	The value of the parameter `inc`
`windowtype`	The value of the parameter `wtype`

Author(s)

Andreas Weingessel

See Also

plot.stft

Examples

```
x<-rnorm(500)
y<-stft(x)
plot(y)
```

---

svm — *Support Vector Machines*

Description

svm is used to train a support vector machine. It can be used to carry out general regression and classification (of nu and epsilon-type), as well as density-estimation. A formula interface is provided.

Usage

```
## S3 method for class 'formula'
svm(formula, data = NULL, ..., subset, na.action =
na.omit, scale = TRUE)
## Default S3 method:
svm(x, y = NULL, scale = TRUE, type = NULL, kernel =
"radial", degree = 3, gamma = if (is.vector(x)) 1 else 1 / ncol(x),
coef0 = 0, cost = 1, nu = 0.5,
class.weights = NULL, cachesize = 40, tolerance = 0.001, epsilon = 0.1,
shrinking = TRUE, cross = 0, probability = FALSE, fitted = TRUE,
..., subset, na.action = na.omit)
```

Arguments

`formula`	a symbolic description of the model to be fit.
`data`	an optional data frame containing the variables in the model. By default the variables are taken from the environment which 'svm' is called from.
`x`	a data matrix, a vector, or a sparse matrix (object of class `Matrix` provided by the **Matrix** package, or of class `matrix.csr` provided by the **SparseM** package, or of class `simple_triplet_matrix` provided by the **slam** package).
`y`	a response vector with one label for each row/component of `x`. Can be either a factor (for classification tasks) or a numeric vector (for regression).
`scale`	A logical vector indicating the variables to be scaled. If `scale` is of length 1, the value is recycled as many times as needed. Per default, data are scaled internally (both `x` and `y` variables) to zero mean and unit variance. The center and scale values are returned and used for later predictions.

`type`	`svm` can be used as a classification machine, as a regression machine, or for novelty detection. Depending of whether `y` is a factor or not, the default setting for `type` is `C-classification` or `eps-regression`, respectively, but may be overwritten by setting an explicit value. Valid options are: • `C-classification` • `nu-classification` • `one-classification` (for novelty detection) • `eps-regression` • `nu-regression`
`kernel`	the kernel used in training and predicting. You might consider changing some of the following parameters, depending on the kernel type. **linear:** $u'v$ **polynomial:** $(\gamma u'v + coef0)^{degree}$ **radial basis:** $e^(- \gamma\|u-v\|^2)$ **sigmoid:** $tanh(\gamma u'v + coef0)$
`degree`	parameter needed for kernel of type `polynomial` (default: 3)
`gamma`	parameter needed for all kernels except `linear` (default: 1/(data dimension))
`coef0`	parameter needed for kernels of type `polynomial` and `sigmoid` (default: 0)
`cost`	cost of constraints violation (default: 1)—it is the 'C'-constant of the regularization term in the Lagrange formulation.
`nu`	parameter needed for `nu-classification`, `nu-regression`, and `one-classification`
`class.weights`	a named vector of weights for the different classes, used for asymmetric class sizes. Not all factor levels have to be supplied (default weight: 1). All components have to be named. Specifying `"inverse"` will choose the weights *inversely* proportional to the class distribution.
`cachesize`	cache memory in MB (default 40)
`tolerance`	tolerance of termination criterion (default: 0.001)
`epsilon`	epsilon in the insensitive-loss function (default: 0.1)
`shrinking`	option whether to use the shrinking-heuristics (default: `TRUE`)
`cross`	if a integer value k>0 is specified, a k-fold cross validation on the training data is performed to assess the quality of the model: the accuracy rate for classification and the Mean Squared Error for regression
`fitted`	logical indicating whether the fitted values should be computed and included in the model or not (default: `TRUE`)
`probability`	logical indicating whether the model should allow for probability predictions.
`...`	additional parameters for the low level fitting function `svm.default`
`subset`	An index vector specifying the cases to be used in the training sample. (NOTE: If given, this argument must be named.)
`na.action`	A function to specify the action to be taken if `NA`s are found. The default action is `na.omit`, which leads to rejection of cases with missing values on any required variable. An alternative is `na.fail`, which causes an error if `NA` cases are found. (NOTE: If given, this argument must be named.)

Details

For multiclass-classification with k levels, k>2, `libsvm` uses the 'one-against-one'-approach, in which k(k-1)/2 binary classifiers are trained; the appropriate class is found by a voting scheme.

`libsvm` internally uses a sparse data representation, which is also high-level supported by the package **SparseM**.

If the predictor variables include factors, the formula interface must be used to get a correct model matrix.

`plot.svm` allows a simple graphical visualization of classification models.

The probability model for classification fits a logistic distribution using maximum likelihood to the decision values of all binary classifiers, and computes the a-posteriori class probabilities for the multi-class problem using quadratic optimization. The probabilistic regression model assumes (zero-mean) laplace-distributed errors for the predictions, and estimates the scale parameter using maximum likelihood.

Value

An object of class `"svm"` containing the fitted model, including:

`SV`	The resulting support vectors (possibly scaled).
`index`	The index of the resulting support vectors in the data matrix. Note that this index refers to the preprocessed data (after the possible effect of `na.omit` and `subset`)
`coefs`	The corresponding coefficients times the training labels.
`rho`	The negative intercept.
`sigma`	In case of a probabilistic regression model, the scale parameter of the hypothesized (zero-mean) laplace distribution estimated by maximum likelihood.
`probA, probB`	numeric vectors of length k(k-1)/2, k number of classes, containing the parameters of the logistic distributions fitted to the decision values of the binary classifiers (1 / (1 + exp(a x + b))).

Note

Data are scaled internally, usually yielding better results.

Parameters of SVM-models usually *must* be tuned to yield sensible results!

Author(s)

David Meyer (based on C/C++-code by Chih-Chung Chang and Chih-Jen Lin)
`<David.Meyer@R-project.org>`

References

- Chang, Chih-Chung and Lin, Chih-Jen:
  LIBSVM: a library for Support Vector Machines
  http://www.csie.ntu.edu.tw/~cjlin/libsvm

- Exact formulations of models, algorithms, etc. can be found in the document:
  Chang, Chih-Chung and Lin, Chih-Jen:
  LIBSVM: a library for Support Vector Machines
  http://www.csie.ntu.edu.tw/~cjlin/papers/libsvm.ps.gz
- More implementation details and speed benchmarks can be found on: Rong-En Fan and Pai-Hsune Chen and Chih-Jen Lin:
  Working Set Selection Using the Second Order Information for Training SVM
  http://www.csie.ntu.edu.tw/~cjlin/papers/quadworkset.pdf

See Also

predict.svm plot.svm tune.svm matrix.csr (in package **SparseM**)

Examples

```
data(iris)
attach(iris)

## classification mode
# default with factor response:
model <- svm(Species ~ ., data = iris)

# alternatively the traditional interface:
x <- subset(iris, select = -Species)
y <- Species
model <- svm(x, y)

print(model)
summary(model)

# test with train data
pred <- predict(model, x)
# (same as:)
pred <- fitted(model)

# Check accuracy:
table(pred, y)

# compute decision values and probabilities:
pred <- predict(model, x, decision.values = TRUE)
attr(pred, "decision.values")[1:4,]

# visualize (classes by color, SV by crosses):
plot(cmdscale(dist(iris[,-5])),
     col = as.integer(iris[,5]),
     pch = c("o","+")[1:150 %in% model$index + 1])

## try regression mode on two dimensions

# create data
x <- seq(0.1, 5, by = 0.05)
y <- log(x) + rnorm(x, sd = 0.2)
```

```
# estimate model and predict input values
m   <- svm(x, y)
new <- predict(m, x)

# visualize
plot(x, y)
points(x, log(x), col = 2)
points(x, new, col = 4)

## density-estimation

# create 2-dim. normal with rho=0:
X <- data.frame(a = rnorm(1000), b = rnorm(1000))
attach(X)

# traditional way:
m <- svm(X, gamma = 0.1)

# formula interface:
m <- svm(~., data = X, gamma = 0.1)
# or:
m <- svm(~ a + b, gamma = 0.1)

# test:
newdata <- data.frame(a = c(0, 4), b = c(0, 4))
predict (m, newdata)

# visualize:
plot(X, col = 1:1000 %in% m$index + 1, xlim = c(-5,5), ylim=c(-5,5))
points(newdata, pch = "+", col = 2, cex = 5)

# weights: (example not particularly sensible)
i2 <- iris
levels(i2$Species)[3] <- "versicolor"
summary(i2$Species)
wts <- 100 / table(i2$Species)
wts
m <- svm(Species ~ ., data = i2, class.weights = wts)
```

---

tune *Parameter Tuning of Functions Using Grid Search*

Description

This generic function tunes hyperparameters of statistical methods using a grid search over supplied parameter ranges.

Usage

```
tune(method, train.x, train.y = NULL, data = list(), validation.x =
     NULL, validation.y = NULL, ranges = NULL, predict.func = predict,
     tunecontrol = tune.control(), ...)
best.tune(...)
```

Arguments

`method`	either the function to be tuned, or a character string naming such a function.
`train.x`	either a formula or a matrix of predictors.
`train.y`	the response variable if `train.x` is a predictor matrix. Ignored if `train.x` is a formula.
`data`	data, if a formula interface is used. Ignored, if predictor matrix and response are supplied directly.
`validation.x`	an optional validation set. Depending on whether a formula interface is used or not, the response can be included in `validation.x` or separately specified using `validation.y`. Only used for bootstrap and fixed validation set (see `tune.control`)
`validation.y`	if no formula interface is used, the response of the (optional) validation set. Only used for bootstrap and fixed validation set (see `tune.control`)
`ranges`	a named list of parameter vectors spanning the sampling space. The vectors will usually be created by `seq`.
`predict.func`	optional predict function, if the standard `predict` behavior is inadequate.
`tunecontrol`	object of class `"tune.control"`, as created by the function `tune.control()`. If omitted, `tune.control()` gives the defaults.
`...`	Further parameters passed to the training functions.

Details

As performance measure, the classification error is used for classification, and the mean squared error for regression. It is possible to specify only one parameter combination (i.e., vectors of length 1) to obtain an error estimation of the specified type (bootstrap, cross-classification, etc.) on the given data set. For convenience, there are several `tune.foo()` wrappers defined, e.g., for `nnet()`, `randomForest()`, `rpart()`, `svm()`, and `knn()`.

Cross-validation randomizes the data set before building the splits which—once created—remain constant during the training process. The splits can be recovered through the `train.ind` component of the returned object.

Value

For `tune`, an object of class `tune`, including the components:

`best.parameters`
: a 1 x k data frame, k number of parameters.

`best.performance`
: best achieved performance.

performances — if requested, a data frame of all parameter combinations along with the corresponding performance results.

train.ind — list of index vectors used for splits into training and validation sets.

best.model — if requested, the model trained on the complete training data using the best parameter combination.

best.tune() returns the best model detected by tune.

Author(s)

David Meyer
<David.Meyer@R-project.org>

See Also

tune.control, plot.tune, tune.svm, tune.wrapper

Examples

```
data(iris)
## tune `svm' for classification with RBF-kernel (default in svm),
## using one split for training/validation set

obj <- tune(svm, Species~., data = iris,
            ranges = list(gamma = 2^(-1:1), cost = 2^(2:4)),
            tunecontrol = tune.control(sampling = "fix")
           )

## alternatively:
## obj <- tune.svm(Species~., data = iris, gamma = 2^(-1:1), cost = 2^(2:4))

summary(obj)
plot(obj)

## tune `knn' using a convenience function; this time with the
## conventional interface and bootstrap sampling:
x <- iris[,-5]
y <- iris[,5]
obj2 <- tune.knn(x, y, k = 1:5, tunecontrol = tune.control(sampling = "boot"))
summary(obj2)
plot(obj2)

## tune `rpart' for regression, using 10-fold cross validation (default)
data(mtcars)
obj3 <- tune.rpart(mpg~., data = mtcars, minsplit = c(5,10,15))
summary(obj3)
plot(obj3)

## simple error estimation for lm using 10-fold cross validation
tune(lm, mpg~., data = mtcars)
```

`tune.control` *Control Parameters for the Tune Function*

Description

Creates an object of class `tune.control` to be used with the `tune` function, containing various control parameters.

Usage

```
tune.control(random = FALSE, nrepeat = 1, repeat.aggregate = mean,
sampling = c("cross", "fix", "bootstrap"), sampling.aggregate = mean,
sampling.dispersion = sd,
cross = 10, fix = 2/3, nboot = 10, boot.size = 9/10, best.model = TRUE,
performances = TRUE, error.fun = NULL)
```

Arguments

`random`	if an integer value is specified, `random` parameter vectors are drawn from the parameter space.
`nrepeat`	specifies how often training shall be repeated.
`repeat.aggregate`	function for aggregating the repeated training results.
`sampling`	sampling scheme. If `sampling = "cross"`, a `cross`-times cross validation is performed. If `sampling = "boot"`, `nboot` training sets of size `boot.size` (part) are sampled (with replacement) from the supplied data. If `sampling = "fix"`, a single split into training/validation set is used, the training set containing a `fix` part of the supplied data. Note that a separate validation set can be supplied via `validation.x` and `validation.y`. It is only used for `sampling = "boot"` and `sampling = "fix"`; in the latter case, `fix` is set to 1.
`sampling.aggregate,sampling.dispersion`	functions for aggregating the training results on the generated training samples (default: mean and standard deviation).
`cross`	number of partitions for cross-validation.
`fix`	part of the data used for training in fixed sampling.
`nboot`	number of bootstrap replications.
`boot.size`	size of the bootstrap samples.
`best.model`	if `TRUE`, the best model is trained and returned (the best parameter set is used for training on the complete training set).
`performances`	if `TRUE`, the performance results for all parameter combinations are returned.
`error.fun`	function returning the error measure to be minimized. It takes two arguments: a vector of true values and a vector of predicted values. If `NULL`, the misclassification error is used for categorical predictions and the mean squared error for numeric predictions.

Value

An object of class "tune.control" containing all the above parameters (either the defaults or the user specified values).

Author(s)

David Meyer
<David.Meyer@R-project.org>

See Also

tune

---

tune.wrapper *Convenience Tuning Wrapper Functions*

Description

Convenience tuning wrapper functions, using tune.

Usage

```
tune.svm(x, y = NULL, data = NULL, degree = NULL, gamma = NULL, coef0 = NULL,
         cost = NULL, nu = NULL, class.weights = NULL, epsilon = NULL, ...)
best.svm(x, tunecontrol = tune.control(), ...)

tune.nnet(x, y = NULL, data = NULL, size = NULL, decay = NULL,
          trace = FALSE, tunecontrol = tune.control(nrepeat = 5),
          ...)
best.nnet(x, tunecontrol = tune.control(nrepeat = 5), ...)

tune.rpart(formula, data, na.action = na.omit, minsplit = NULL,
           minbucket = NULL, cp = NULL, maxcompete = NULL, maxsurrogate = NULL,
           usesurrogate = NULL, xval = NULL, surrogatestyle = NULL, maxdepth =
           NULL, predict.func = NULL, ...)
best.rpart(formula, tunecontrol = tune.control(), ...)

tune.randomForest(x, y = NULL, data = NULL, nodesize = NULL,
                  mtry = NULL, ntree = NULL, ...)
best.randomForest(x, tunecontrol = tune.control(), ...)

tune.knn(x, y, k = NULL, l = NULL, ...)
```

Arguments

formula, x, y, data
: formula and data arguments of function to be tuned.

predict.func
: predicting function.

na.action
: function handling missingness.

minsplit, minbucket, cp, maxcompete, maxsurrogate, usesurrogate, xval, surrogatestyle, maxdepth
: rpart parameters.

degree, gamma, coef0, cost, nu, class.weights, epsilon
: svm parameters.

k, l
: knn parameters.

mtry, nodesize, ntree
: randomForest parameters.

size, decay, trace
: parameters passed to nnet.

tunecontrol
: object of class "tune.control" containing tuning parameters.

...
: Further parameters passed to tune.

Details

For examples, see the help page of tune().

Value

tune.foo() returns a tuning object including the best parameter set obtained by optimizing over the specified parameter vectors. best.foo() directly returns the best model, i.e. the fit of a new model using the optimal parameters found by tune.foo.

Author(s)

David Meyer
<David.Meyer@R-project.org>

See Also

tune

---

write.svm *Write SVM Object to File*

Description

This function exports an SVM object (trained by svm) to two specified files. One is in the format that the function 'svm_load_model' of libsvm can read. The other is for scaling data, containing a data with centers and scales for all variables.

Usage

```
write.svm(object, svm.file = "Rdata.svm",
          scale.file = "Rdata.scale", yscale.file = "Rdata.yscale")
```

Arguments

`object`	Object of class "svm", created by `svm`.
`svm.file`	filename to export the svm object to.
`scale.file`	filename to export the scaling data of the explanatory variables to.
`yscale.file`	filename to export the scaling data of the dependent variable to, if any.

Details

This function is useful when SVM models trained in R shall be used in other environments. The SVM model is saved in the standard format of libsvm. The scaling data are written to a separate file because scaling data are not included in the standard format of libsvm. The format of the scaling data file is a n times 2 matrix: the n-th row corresponds to the n-th dimension of the data, the columns being formed of the corresponding mean and scale. If scaling information for the dependent variable exists (in case of regression models), it is stored in yet another file (1 times 2 matrix).

Author(s)

Tomomi TAKASHINA (based on 'predict.svm' by David Meyer) <t.takashina@computer.org>

See Also

`svm`

Examples

```
data(iris)
attach(iris)

## classification mode
# default with factor response:
model <- svm (Species~., data=iris)

# export SVM object to file
write.svm(model, svm.file = "iris-classifier.svm", scale.file = "iris-classifier.scale")

# read scale file
# the n-th row is corresponding to n-th dimension. The 1st column contains the
# center value, the 2nd column is the scale value.

read.table("iris-classifier.scale")
```

Index