Package 'mt'

February 12, 2024

Version 2.0-1.20

Date 2024-02-12

Title Metabolomics Data Analysis Toolbox

Author Wanchang Lin

Maintainer Wanchang Lin <wanchanglin@hotmail.com>

Description Functions for metabolomics data analysis: data preprocessing, orthogonal signal correction, PCA analysis, PCA-DA analysis, PLS-DA analysis, classification, feature selection, correlation analysis, data visualisation and re-sampling strategies.

Depends R (>= 3.0.0)

Imports methods, MASS, class, e1071, randomForest, pls, ellipse, lattice, latticeExtra

Encoding UTF-8

License GPL (>= 2)

URL https://github.com/wanchanglin/mt

BugReports https://github.com/wanchanglin/mt/issues

LazyLoad yes

LazyData yes

ZipData No

NeedsCompilation no

Repository CRAN

Date/Publication 2024-02-12 12:40:02 UTC

R topics documented:

`abr1` *abr1 Data*

Description

An FIE-MS data.

Usage

```
data(abr1)
```

Details

`abr1` is an FIE-MS data matrices developed from analysis of samples representing a time course of pathogen attack in a model plant species (Brachypodium distachyon). The data was developed in a single batch with all samples randomised using a Thermo LTQ linear ion trap. Both positive and negative ion mode are given (`abr1$pos` and `abr1$neg`).

Value

A list with the following elements:

`fact`	A data frame containing experimental meta-data.
`pos`	A data frame for positive data with 120 observations and 2000 variables.
`neg`	A data frame for negative data with 120 observations and 2000 variables.

Examples

```
# Load data set
data(abr1)

# Select data set
dat <- abr1$neg

# number of observations and variables
dim(dat)

# Transform data
dat.log   <- preproc(dat, method = "log")
dat.sqrt  <- preproc(dat, method = "sqrt")
dat.asinh <- preproc(dat, method = "asinh")

op <- par(mfrow=c(2,2), pch=16)
matplot(t(dat),main="Original",type="l",col="blue",
     ylab="Intensity")
matplot(t(dat.log),main="Log",type="l",col="green",
     ylab="Intensity")
matplot(t(dat.sqrt),main="Sqrt",type="l",col="red",
     ylab="Intensity")
matplot(t(dat.asinh),main="ArcSinh)",type="l",col="black",
     ylab="Intensity")
par(op)
mtext("Data set", line=2.5, font=3, cex=1.5)
```

---

accest *Estimate Classification Accuracy By Resampling Method*

Description

Estimate classification accuracy rate by resampling method.

Usage

```
accest(dat, ...)

## Default S3 method:
accest(dat, cl, method, pred.func=predict,pars = valipars(),
       tr.idx = NULL, ...)

## S3 method for class 'formula'
accest(formula, data = NULL, ..., subset, na.action = na.omit)

aam.cl(x,y,method, pars = valipars(),...)

aam.mcl(x,y,method, pars = valipars(),...)
```

Arguments

formula	A formula of the form groups ~ x1 + x2 + ... 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 formula are preferentially to be taken.
dat,x	A matrix or data frame containing the explanatory variables if no formula is given as the principal argument.
cl,y	A factor specifying the class for each observation if no formula principal argument is given.
method	Classification method whose accuracy rate is to be estimated, such as randomForest, svm, knn and lda. For details, see note below. Either a function or a character string naming the function to be called.
pred.func	Predict method (default is predict). Either a function or a character string naming the function to be called.
pars	A list of parameters using by the resampling method such as *Leave-one-out cross-validation*, *Cross-validation*, *Bootstrap* and *Randomised validation (holdout)*. See valipars for details.
tr.idx	User defined index of training samples. Can be generated by trainind.
...	Additional parameters to method.
subset	Optional vector, specifying a subset of observations to be used.
na.action	Function which indicates what should happen when the data contains NA's, defaults to na.omit.

Details

The accuracy rates of classification are estimated by techniques such as *Random Forest*, *Support Vector Machine*, *k-Nearest Neighbour Classification* and *Linear Discriminant Analysis* based on resampling methods, including *Leave-one-out cross-validation*, *Cross-validation*, *Bootstrap* and *Randomised validation (holdout)*.

Value

accest returns an object including the components:

method	Classification method used.
acc	Overall accuracy rate.
acc.iter	Average accuracy rate for each iteration.
acc.all	Accuracy rate for each iteration and replication.
auc	Overall area under receiver operating curve (AUC).
auc.iter	Average AUC for each iteration.
auc.all	AUC for each iteration and replication.
mar	Overall prediction margin.
mar.iter	Average prediction margin for each iteration.

mar.all	Prediction margin for each iteration and replication.
err	Overall error rate.
err.iter	Average error rate for each iteration.
err.all	Error rate for each iteration and replication.
sampling	Sampling scheme used.
niter	Number of iteration.
nreps	Number of replications in each iteration if resampling is not loocv.
conf	Overall confusion matrix.
res.all	All results which can be further processed.
acc.boot	A list of bootstrap accuracy such as .632 and .632+ if the resampling method is bootstrap.

aam.cl returns a vector with acc (accuracy), auc(area under ROC curve) and mar(class margin).

aam.mcl returns a matrix with columns of acc (accuracy), auc(area under ROC curve) and mar(class margin).

Note

The accest can take any classification models if their argument format is model(formula, data, subset, na.action, ...) and their corresponding method predict.model(object, newdata, ...) can either return the only predicted class label or a list with a component called class, such as lda and pcalda.

If classifier method provides posterior probabilities, the prediction margin mar will be generated, otherwise NULL.

If classifier method provides posterior probabilities and the classification is for two-class problem, auc will be generated, otherwise NULL.

aam.cl is a wrapper function of accest, returning accuracy rate, AUC and classification margin. aam.mcl accepts multiple classifiers in one running.

Author(s)

Wanchang Lin

See Also

binest, maccest, valipars, trainind, classifier

Examples

```
# Iris data
data(iris)
# Use KNN classifier and bootstrap for resampling
acc <- accest(Species~., data = iris, method = "knn",
              pars = valipars(sampling = "boot",niter = 2, nreps=5))
acc
summary(acc)
```

```
acc$acc.boot

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

## ------------------------------------------------------------------------
# Random Forest with 5-fold stratified cv
pars   <- valipars(sampling = "cv",niter = 4, nreps=5, strat=TRUE)
tr.idx <- trainind(y,pars=pars)
acc1   <- accest(x, y, method = "randomForest", pars = pars, tr.idx=tr.idx)
acc1
summary(acc1)
# plot the accuracy in each iteration
plot(acc1)

## ------------------------------------------------------------------------
# Forensic Glass data in chap.12 of MASS
data(fgl, package = "MASS")    # in MASS package
# Randomised validation (holdout) of SVM for fgl data
acc2 <- accest(type~., data = fgl, method = "svm", cost = 100, gamma = 1,
              pars = valipars(sampling = "rand",niter = 10, nreps=4,div = 2/3) )

acc2
summary(acc2)
# plot the accuracy in each iteration
plot(acc2)

## ------------------------------------------------------------------------
## Examples of amm.cl and aam.mcl
aam.1 <- aam.cl(x,y,method="svm",pars=pars)
aam.2 <- aam.mcl(x,y,method=c("svm","randomForest"),pars=pars)

## If use two classes, AUC will be calculated
idx <- (y == "setosa")
aam.3 <- aam.cl(x[!idx,],factor(y[!idx]),method="svm",pars=pars)
aam.4 <- aam.mcl(x[!idx,],factor(y[!idx]),method=c("svm","randomForest"),pars=pars)
```

`binest` *Binary Classification*

Description

Binary classification.

Usage

```
binest(dat, cl, choices = NULL, method, pars=valipars(),...)
```

Arguments

dat	A matrix or data frame containing the explanatory variables.
cl	A factor specifying the class for each observation.
choices	The vector or list of class labels to be chosen for binary classification. For details, see dat.sel.
method	Classification method to be used. For details, see accest.
pars	A list of parameters of the resampling method. For details, see valipars.
...	Additional parameters to method.

Value

A list with components:

com	A matrix of combination of the binary class labels.
acc	A table of classification accuracy for the binary combination in each iteration.
method	Classification method used.
sampling	Sampling scheme used.
niter	Number of iterations.
nreps	Number of replications in each iteration if resampling is not loocv.

Author(s)

Wanchang Lin

See Also

accest, valipars, dat.sel

Examples

```
# iris data set
data(iris)
dat <- subset(iris, select = -Species)
cl  <- iris$Species

## PCALDA with cross-validation
pars    <- valipars(sampling="cv",niter = 6, nreps = 5)
binpcalda <- binest(dat,cl,choices=c("setosa"), method="pcalda", pars = pars)

## SVM with leave-one-out cross-validation. SVM kernel is 'linear'.
pars   <- valipars(sampling="loocv")
binsvm <- binest(dat,cl,choices=c("setosa","virginica"), method="svm",
                 pars = pars, kernel="linear")

## randomForest with bootstrap
pars  <- valipars(sampling="boot",niter = 5, nreps = 5)
binrf <- binest(dat,cl,choices=c("setosa","virginica"),
                method="randomForest", pars = pars)
```

```
## KNN with randomised validation. The number of neighbours is 3.
pars   <- valipars(sampling="rand",niter = 5, nreps = 5)
binknn <- binest(dat,cl,choices = list(c("setosa","virginica"),
                                       c("virginica","versicolor")),
                 method="knn",pars = pars, k = 3)
```

`boot.err` *Calculate .632 and .632+ Bootstrap Error Rate*

Description

Calculate .632 bootstrap and .632 plus bootstrap error rate.

Usage

```
boot.err(err, resub)
```

Arguments

`err`	Average error rate of bootstrap samples.
`resub`	A list including apparent error rate, class label and the predicted class label of the original training data (not resampled training data). Can be generated by `classifier`.

Value

A list with the following components:

`ae`	Apparent error rate.
`boot`	Average error rate of bootstrap samples(Same as `err`)
`b632`	.632 bootstrap error rate.
`b632p`	.632 plus bootstrap error rate.

Author(s)

Wanchang Lin

References

Witten, I. H. and Frank, E. (2005) *Data Mining - Practical Machine Learning and Techniques*. Elsevier.

Efron, B. and Tibshirani, R. (1993) *An Introduction to the Bootstrap*. Chapman & Hall.

Efron, B. and Tibshirani, R. (1997) Improvements on cross-validation: the .632+ bootstrap method. *Journal of the American Statistical Association*, **92**, 548-560.

See Also

classifier

Examples

```
## iris data set
data(iris)
x      <- subset(iris, select = -Species)
y      <- iris$Species

## 10 bootstrap training samples
pars   <- valipars(sampling = "boot", niter = 1, nreps = 10)
tr.idx <- trainind(y, pars=pars)[[1]]

## bootstrap error rate
err <- sapply(tr.idx, function(i){
  pred <- classifier(x[i,,drop = FALSE],y[i],x[-i,,drop = FALSE],y[-i],
                     method = "knn")$err
})

## average bootstrap error rate
err <- mean(err)

## apparent error rate
resub  <- classifier(x,y,method = "knn")

##
err.boot <- boot.err(err, resub)
```

---

boxplot.frankvali *Boxplot Method for Class 'frankvali'*

Description

Boxplot method for error rate of each feature subset.

Usage

```
## S3 method for class 'frankvali'
boxplot(x,  ...)
```

Arguments

x	An object of class frankvali.
...	Additional arguments to the plot, such as main, xlab and ylab.

Details

This function is a method for the generic function `boxplot()` for class `frankvali`. It plots the error rate of each feature subset.

Value

Returns boxplot of class `frankvali`.

Author(s)

Wanchang Lin

See Also

frankvali

Examples

```
data(abr1)
dat <- abr1$pos[,110:500]

x   <- preproc(dat, method="log10")
y   <- factor(abr1$fact$class)

dat <- dat.sel(x, y, choices=c("1","2"))
x.1 <- dat[[1]]$dat
y.1 <- dat[[1]]$cls

pars <- valipars(sampling="cv",niter=2,nreps=4)
res  <- frankvali(x.1,y.1,fs.method = "fs.rfe",fs.len = "power2",
                  cl.method = "knn",pars = pars)
res
summary(res)
boxplot(res)
```

`boxplot.maccest` *Boxplot Method for Class 'maccest'*

Description

Boxplot method for the accuracy rate of each classifier.

Usage

```
## S3 method for class 'maccest'
boxplot(x,  ...)
```

Arguments

x	An object of class maccest.
...	Additional arguments to the plot, such as main, xlab and ylab.

Details

This function is a method for the generic function boxplot() for class maccest. It plots the accurary rate for each classifier.

Value

Returns boxplot of class maccest.

Author(s)

Wanchang Lin

See Also

maccest, plot.maccest

Examples

```
# Iris data
data(iris)
x      <- subset(iris, select = -Species)
y      <- iris$Species

method <- c("randomForest","svm","knn")
pars   <- valipars(sampling="cv", niter = 2, nreps=5)
tr.idx <- trainind(y, pars=pars)
res    <- maccest(x, y, method=method, pars=pars,
                  comp="anova",kernel="linear")

res
boxplot(res)
```

---

cl.rate *Assess Classification Performances*

Description

Assess classification performances.

Usage

```
cl.rate(obs, pre)
cl.perf(obs, pre, pos=levels(as.factor(obs))[2])
cl.roc(stat, label, pos=levels(as.factor(label))[2], plot=TRUE, ...)
cl.auc(stat, label, pos=levels(as.factor(label))[2])
```

Arguments

obs	Factor or vector of observed class.
pre	Factor or vector of predicted class.
stat	Factor or vector of statistics for positives/cases.
label	Factor or vector of label for categorical data.
pos	Characteristic string for positive.
plot	Logical flag indicating whether ROC should be plotted.
...	Further arguments for plotting.

Details

cl.perf gets the classification performances such as accuracy rate and false positive rate. cl.roc computes receiver operating characteristics (ROC). cl.auc calculates area under ROC curve. Three functions are only for binary class problems.

Value

cl.rate returns a list with components:

acc	Accuracy rate of classification.
err	Error rate of classification.
con.mat	Confusion matrix.
kappa	Kappa Statistics.

cl.perf returns a list with components:

acc	Accuracy rate
tpr	True positive rate
fpr	False positive rate
sens	Sensitivity
spec	Specificity
con.mat	Confusion matrix.
kappa	Kappa Statistics.
positive	Positive level.

cl.roc returns a list with components:

perf	A data frame of acc, tpr,fpr,sens, spec and cutoff (thresholds).
auc	Area under ROC curve
positive	Positive level.

cl.auc returns a scalar value of AUC.

Note

AUC varies between 0.5 and 1.0 for sensible models; the higher the better. If it is less than 0.5, it should be corrected by `1 - AUC`. Or re-run it by using `1 - stat`.

Author(s)

Wanchang Lin

References

Fawcett, F. (2006) *An introduction to ROC analysis*. *Pattern Recognition Letters*. vol. 27, 861-874.

Examples

```
## Measurements of Forensic Glass Fragments
library(MASS)
data(fgl, package = "MASS")    # in MASS package
dat <- subset(fgl, grepl("WinF|WinNF",type))
## dat <- subset(fgl, type %in% c("WinF", "WinNF"))
x   <- subset(dat, select = -type)
y   <- factor(dat$type)

## construct train and test data
idx   <- sample(1:nrow(x), round((2/3)*nrow(x)), replace = FALSE)
tr.x  <- x[idx,]
tr.y  <- y[idx]
te.x  <- x[-idx,]
te.y  <- y[-idx]

model <- lda(tr.x, tr.y)

## predict the test data results
pred  <- predict(model, te.x)

## classification performances
obs <- te.y
pre <- pred$class
cl.rate(obs, pre)
cl.perf(obs, pre, pos="WinNF")
## change positive as "WinF"
cl.perf(obs, pre, pos="WinF")

## ROC and AUC
pos  <- "WinNF"            ## or "WinF"
stat <- pred$posterior[,pos]
## levels(obs) <- c(0,1)

cl.auc (stat,obs, pos=pos)
cl.roc (stat,obs, pos=pos)

## test examples for ROC and AUC
```

```
label <- rbinom(30,size=1,prob=0.2)
stat  <- rnorm(30)
cl.roc(stat,label, pos=levels(factor(label))[2],plot = TRUE)
cl.auc(stat,label,pos=levels(factor(label))[2])

## if auc is less than 0.5, it should be adjusted by 1 - auc.
## Or re-run them:
cl.roc(1 - stat,label, pos=levels(factor(label))[2],plot = TRUE)
cl.auc(1 - stat,label,pos=levels(factor(label))[2])
```

`classifier` *Wrapper Function for Classifiers*

Description

Wrapper function for classifiers. The classification model is built up on the training data and error estimation is performed on the test data.

Usage

```
classifier(dat.tr, cl.tr, dat.te=NULL, cl.te=NULL, method,
           pred.func=predict,...)
```

Arguments

`dat.tr`	A data frame or matrix of training data. The classification model are built on it.
`cl.tr`	A factor or vector of training class.
`dat.te`	A data frame or matrix of test data. Error rates are calculated on this data set.
`cl.te`	A factor or vector of test class.
`method`	Classification method to be used. Any classification methods can be employed if they have method `predict` (except `knn`) with output of predicted class label or one component with name of `class` in the returned list, such as `randomForest`, `svm`, `knn` and `lda`. Either a function or a character string naming the function to be called
`pred.func`	Predict method (default is `predict`). Either a function or a character string naming the function to be called.
`...`	Additional parameters to `method`.

Value

A list including components:

`err`	Error rate of test data.
`cl`	The original class of test data.
`pred`	The predicted class of test data.

posterior	Posterior probabilities for the classes if method provides posterior output.
acc	Accuracy rate of classification.
margin	The margin of predictions from classifier method if it provides posterior output. The margin of a data point is defined as the proportion of probability for the correct class minus maximum proportion of probabilities for the other classes. Positive margin means correct classification, and vice versa. This idea come from package **randomForest**. For more details, see margin.
auc	The area under receiver operating curve (AUC) if classifier method produces posterior probabilities and the classification is for two-class problem.

Note

The definition of margin is based on the posterior probabilities. Classifiers, such as randomForest, svm, lda, qda, pcalda and plslda, do output posterior probabilities. But knn does not.

Author(s)

Wanchang Lin

See Also

accest, maccest

Examples

```
data(abr1)
dat <- preproc(abr1$pos[,110:500], method="log10")
cls <- factor(abr1$fact$class)

## tmp <- dat.sel(dat, cls, choices=c("1","2"))
## dat <- tmp[[1]]$dat
## cls <- tmp[[1]]$cls

idx <- sample(1:nrow(dat), round((2/3)*nrow(dat)), replace = FALSE)
## constrcuct train and test data
train.dat  <- dat[idx,]
train.cl   <- cls[idx]
test.dat   <- dat[-idx,]
test.cl    <- cls[-idx]

## estimates accuracy
res <- classifier(train.dat, train.cl, test.dat, test.cl,
                  method="randomForest")
res
## get confusion matrix
cl.rate(obs=res$cl, res$pred)   ## same as: cl.rate(obs=test.cl, res$pred)

## Measurements of Forensic Glass Fragments
data(fgl, package = "MASS")    # in MASS package
dat <- subset(fgl, grepl("WinF|WinNF",type))
```

```
## dat <- subset(fgl, type %in% c("WinF", "WinNF"))
x   <- subset(dat, select = -type)
y   <- factor(dat$type)

## construct train and test data
idx   <- sample(1:nrow(x), round((2/3)*nrow(x)), replace = FALSE)
tr.x  <- x[idx,]
tr.y  <- y[idx]
te.x  <- x[-idx,]
te.y  <- y[-idx]

res.1 <- classifier(tr.x, tr.y, te.x, te.y, method="svm")
res.1
cl.rate(obs=res.1$cl, res.1$pred)

## classification performance for the two-class case.
pos <- "WinF"                              ## select positive level
cl.perf(obs=res.1$cl, pre=res.1$pred, pos=pos)
## ROC and AUC
cl.roc(stat=res.1$posterior[,pos],label=res.1$cl, pos=pos)
```

---

cor.util *Correlation Analysis Utilities*

Description

Functions to handle correlation analysis on data set.

Usage

```
cor.cut(mat,cutoff=0.75,abs.f = FALSE,
        use = "pairwise.complete.obs", method = "pearson",...)

cor.hcl(mat, cutoff=0.75, use = "pairwise.complete.obs",
        method = "pearson",fig.f=TRUE, hang=-1,
        horiz = FALSE, main = "Cluster Dendrogram",
        ylab = ifelse(!horiz, "1 - correlation",""),
        xlab = ifelse(horiz, "1 - correlation",""),...)

cor.heat(mat, use = "pairwise.complete.obs", method = "pearson",
         dend = c("right", "top", "none"),...)

corrgram.circle(co,
                col.regions = colorRampPalette(c("red", "white", "blue")),
                scales = list(x = list(rot = 90)), ...)

corrgram.ellipse(co,label=FALSE,
                 col.regions = colorRampPalette(c("red", "white", "blue")),
```

```
                  scales = list(x = list(rot = 90)), ...)

cor.heat.gram(mat.1, mat.2, use = "pairwise.complete.obs",
              method = "pearson", main="Heatmap of correlation",
              cex=0.75, ...)

hm.cols(low = "green", high = "red", n = 123)
```

Arguments

mat, mat.1, mat.2	A data frame or matrix. It should be noticed that mat.1 and mat.2 must have the same number of row.
cutoff	A scalar value of threshold.
abs.f	Logical flag indicating whether the absolute values should be used.
fig.f	Logical flag indicating whether the dendrogram of correlation matrix should be plotted.
hang	The fraction of the plot height by which labels should hang below the rest of the plot. A negative value will cause the labels to hang down from 0. See plot.hclust.
horiz	Logical indicating if the dendrogram should be drawn *horizontally* or not.
main, xlab, ylab	Graphical parameters, see plot.default.
dend	Character string indicating whether to draw 'right', 'top' or 'none' dendrograms
.	
use	Argument for cor. An optional character string giving a method for computing covariances in the presence of missing values. This must be (an abbreviation of) one of the strings "everything", "all.obs", "complete.obs", "na.or.complete", or "pairwise.complete.obs".
method	Argument for cor. A character string indicating which correlation coefficient (or covariance) is to be computed. One of "pearson", "kendall", or "spearman", can be abbreviated.
co	Correlation matrix
label	A logical value indicating whether the correlation coefficient should be plotted.
...	Additional parameters to **lattice**.
col.regions	Color vector to be used
scales	A list determining how the x- and y-axes (tick marks and labels) are drawn. More details, see xyplot.
cex	A numeric multiplier to control character sizes for axis labels
.	
low	Colour for low value
high	Colour for high value
n	The number of colors (>= 1) to be in the palette

Details

`cor.cut` returns the pairs with correlation coefficient larger than `cutoff`.

`cor.hcl` computes hierarchical cluster analysis based on correlation coefficient. For other graphical parameters, see `plot.dendrogram`.

`cor.heat` display correlation heatmap using **lattice**.

`corrgram.circle` and `corrgram.ellipse` display corrgrams with circle and ellipse. The functions are modified from codes given in Deepayan Sarkar's `Lattice: Multivariate Data Visualization with R, 13.3.3 Corrgrams as customized level plots, pp:238-241`.

`cor.heat.gram` handles the correlation of two data sets which have the same row number. The best application is correlation between MS data (metabolites) and meta/clinical data.

`hm.cols` creates a vector of n contiguous colors for heat map.

Value

`cor.cut` returns a data frame with three columns, in which the first and second columns are variable names and their correlation (lager than cutoff) are given in the third column.

`cor.hcl` returns a list with components of each cluster group and all correlation coefficients.

`cor.heat` returns an object of class "trellis".

`corrgram.circle` returns an object of class "trellis".

`corrgram.ellipse` returns an object of class "trellis".

`cor.heat.gram` returns a list including the components:

- `cor.heat`: An object of class "trellis" for correlation heatmap ordered by the hierarchical clustering.
- `cor.gram`: An object of class "trellis" for corrgrams with circle ordered by the hierarchical clustering.
- `cor.short`: A matrix of correlation coefficient in short format.
- `cor.long`: A matrix of correlation coefficient in long format.

Author(s)

Wanchang Lin

References

Michael Friendly (2002). *Corrgrams: Exploratory displays for correlation matrices*. The American Statistician, 56, 316–324.

D.J. Murdoch, E.D. Chow (1996). *A graphical display of large correlation matrices*. The American Statistician, 50, 178–180.

Deepayan Sarkar (2008). *Lattice: Multivariate Data Visualization with R*. Springer.

Examples

```
data(iris)
cor.cut(iris[,1:4],cutoff=0.8, use="pairwise.complete.obs")
cor.hcl(iris[,1:4],cutoff=0.75,fig.f = TRUE)
ph <- cor.heat(iris[,1:4], dend="top")
ph
update(ph, scales = list(x = list(rot = 45)))

## change heatmap color scheme
cor.heat(iris[,1:4], dend="right", xlab="", ylab="",
  col.regions = colorRampPalette(c("green", "black", "red")))

## or use hm.cols
cor.heat(iris[,1:4], dend="right", xlab="", ylab="", col.regions = hm.cols())

## prepare data set
data(abr1)
cls <- factor(abr1$fact$class)
dat <- preproc(abr1$pos[,110:1930], method="log10")

## feature selection
res <- fs.rf(dat,cls)
## take top 20 features
fs  <- res$fs.order[1:20]
## construct the data set for correlation analysis
mat <- dat[,fs]

cor.cut(mat,cutoff=0.9)
ch <- cor.hcl(mat,cutoff=0.75,fig.f = TRUE, xlab="Peaks")
## plot dendrogram horizontally with coloured labels.
ch <- cor.hcl(mat,cutoff=0.75,fig.f = TRUE, horiz=TRUE,center=TRUE,
              nodePar = list(lab.cex = 0.6, lab.col = "forest green", pch = NA),
              xlim=c(2,0))

names(ch)
cor.heat(mat,dend="right")
cor.heat(mat,dend="right",col.regions = colorRampPalette(c("yellow", "red")))

## use corrgram with order by the hierarchical clustering
co  <- cor(mat, use="pairwise.complete.obs")
ord <- order.dendrogram(as.dendrogram(hclust(as.dist(1-co))))
corrgram.circle(co[ord,ord], main="Corrgrams with circle")
corrgram.ellipse(co[ord,ord], label = TRUE, main = "Corrgrams with circle",
                 col.regions = hm.cols())

## if without ordering
corrgram.circle(co, main="Corrgrams with circle")

## example of cor.heat.gram
fs.1  <- res$fs.order[21:50]
mat.1 <- dat[,fs.1]
```

```
res.cor <-
  cor.heat.gram(mat, mat.1, main="Heatmap of correlation between mat.1 and mat.2")
names(res.cor)
res.cor$cor.heat
res.cor$cor.gram
```

`dat.sel` *Generate Pairwise Data Set*

Description

Generate index or data set of pairwise combination based on class labels.

Usage

```
combn.pw(cls, choices = NULL)

dat.sel(dat, cls, choices = NULL)
```

Arguments

`dat`	A data frame or matrix of data set.
`cls`	A factor or vector of class labels or categorical data.
`choices`	The vector or list of class labels to be chosen for binary combination.

Details

If `choices` is `NULL`, all binary combinations will be computed. If `choices` has one class label, the comparisons between this one and any other classes will be calculated. If `choices` has more than two classes, all binary combinations in `choices` will be generated. For details, see `examples` below.

Value

`combn.pw` returns a data frame of index (logical values).

`dat.set` returns a list of list with components:

`dat`	Pairwise data set.
`cls`	Pairwise class label.

Author(s)

Wanchang Lin

See Also

Applications of `dat.sel` in `pca_plot_wrap`, `lda_plot_wrap` and `pls_plot_wrap`.

Examples

```
data(iris)
x <- subset(iris, select = -Species)
y <- iris$Species

## generate data set with class "setosa" and "virginica"
binmat.1 <- dat.sel(x,y,choices=c("setosa","virginica"))
names(binmat.1)

## generate data sets for "setosa" vs other classes. These are:
## "setosa" and "versicolor", "setosa" and "virginica".
binmat.2 <- dat.sel(x,y,choices=c("setosa"))
names(binmat.2)

## generate data set with combination of each class. These are:
## "setosa" and "versicolor", "setosa" and "virginica",
## "versicolor" and "virginica"
binmat.3 <- dat.sel(x,y,choices= NULL)
names(binmat.3)

data(abr1)
cls <- factor(abr1$fact$class)
dat <- preproc(abr1$pos, method="log")

## There are some examples of 'choices'
choices <- c("2")
choices <- c("2","3","4")
choices <- list(c("2","3"),c("4","5"))
choices <- NULL
idx <- combn.pw(cls,choices=choices)

dat.pw <- dat.sel(dat, cls,choices=choices)
```

`data.visualisation` *Grouped Data Visualisation by PCA, MDS, PCADA and PLSDA*

Description

Grouped data visualisation by PCA, MDS, PCADA and PLSDA.

Usage

```
pca_plot_wrap(data.list,title="plotting",...)

mds_plot_wrap(data.list,method="euclidean",title="plotting",...)

lda_plot_wrap(data.list,title="plotting",...)
```

```
lda_plot_wrap.1(data.list,title="plotting",...)

pls_plot_wrap(data.list,title="plotting",...)
```

Arguments

data.list	A two-layer list structure, in which the second layer include a data frame called dat and a factor of class label called cls. Noticed that names of the first layer of data.list should be given. data.list can be produced by dat.sel.
method	The distance measure to be used. This must be one of "euclidean", "maximum", "manhattan", "canberra", "binary" or "minkowski". Any unambiguous substring can be given. It is only for mds_plot_wrap.
title	A part of title string for plotting.
...	Further arguments to lattice. See corresponding entry in xyplot for non-trivial details of lattice. One argument is ep: an integer flag for ellipse. 1 and 2 for plotting overall and group ellipse, respectively. Otherwise, none. For details, see panel.elli.1.

Value

mds_plot_wrap returns a handle for MDS plot.

All other four functions return a list with components: the first one is an object of class "trellis" for data visualisation; the second one is also an object of class "trellis" but plotting the corresponding variables, PCs (principal components), LDs (linear discrimniants) and LCs (latent components); and the third one is a matrix of these variables.

Note

There is a slight differences between lda_plot_wrap.1 and lda_plot_wrap. The former plots the two-class grouped data, which has one linear discriminant (LD1), with strip plot. The later plots the two-class data by LD1 vs LD2 which is identical to LD1. Hence lda_plot_wrap is more general and can be applied to fusion of two and more class data sets.

Author(s)

Wanchang Lin

See Also

pcaplot, mdsplot, plot.pcalda, plot.plsc, dat.sel, grpplot, panel.elli.1.

Examples

```
data(iris)
x <- subset(iris, select = -Species)
y <- iris$Species
## generate data list by dat.sel
iris.pw <- dat.sel(x,y,choices=NULL)
names(iris.pw)
```

```
pca.p <- pca_plot_wrap(iris.pw, ep=2)
pca.p[[1]]     ## visualised by PCA
pca.p[[2]]     ## plot PCA variables
pca.p[[3]]     ## matrix of PCA variables

mds.p <- mds_plot_wrap(iris.pw)
mds.p

pls.p  <- pls_plot_wrap(iris.pw)
pls.p[[1]]
pls.p[[2]]
pls.p[[3]]

lda.p <- lda_plot_wrap.1(iris.pw)
lda.p[[1]]
lda.p[[2]]
lda.p[[3]]
lda_plot_wrap(iris.pw)$lda.p

## only plot iris data
ph <- pca_plot_wrap(list(list(dat=x, cls=y)))$pca.p
## Not given data names
ph
update(ph, strip=FALSE)       ## strip is an argument of lattice

tmp <- list(iris.dat=list(dat=x, cls=y))
pca_plot_wrap(tmp)$pca.p
pca_plot_wrap(tmp,strip=FALSE)$pca.p
pls_plot_wrap(tmp,strip=FALSE)$pls.p
lda_plot_wrap(tmp,strip=FALSE)$lda.p

data(abr1)
cls <- factor(abr1$fact$class)
dat <- preproc(abr1$pos, method="log")
## pair-wise data set
dat.pw <- dat.sel(dat, cls,choices=c("2","3","4"))

## add mult-class
idx <- grep("2|3|4",cls)
cls.234 <- factor(cls[idx])
dat.234 <- dat[idx,,drop = FALSE]

## combine all
dat.tmp <- c(dat.pw,
             "2~3~4"=list(list(dat=dat.234,cls=cls.234)),
             all=list(list(dat=dat, cls=cls)))

## PCA
ph <- pca_plot_wrap(dat.tmp, title="abr1", par.strip.text = list(cex=0.75),
                    scales=list(cex =.75,relation="free"), ep=2)
## See function grpplot for usage of ep.
ph[[1]]
```

```
ph[[2]]

##PLSDA
ph <- pls_plot_wrap(dat.tmp, title="abr1", par.strip.text = list(cex=0.75),
                    scales=list(cex =.75,relation="free"), ep=2)
ph[[1]]
ph[[2]]

## PCADA
ph <- lda_plot_wrap(dat.tmp, title="abr1", par.strip.text = list(cex=0.75),
                    scales=list(cex =.75,relation="free"))
ph[[1]]
ph[[2]]
```

`df.util` *Summary Utilities*

Description

Functions to summarise data.

Usage

```
df.summ(dat, method=vec.summ,...)

vec.summ(x)

vec.summ.1(x)
```

Arguments

`dat`	A data frame or matrix of data set.
`x`	A vector value.
`method`	Summary method such as `vec.summ` and `vec.summ.1`. For user-defined methods, see examples below.
`...`	Additional parameters to `method` function.

Value

`df.summ` returns a summarised data frame.

`vec.summ` returns an vector of number of variables (exclusing NAs), minimum, mean, median, maximum and standard derivation.

`vec.summ.1` returns an vector of number of variables (exclusing NAs), mean, median, 95% confidence interval of median, IQR and standard derivation.

Author(s)

Wanchang Lin

Examples

```
data(abr1)
dat <- (abr1$pos)[,110:150]
cls <- factor(abr1$fact$class)

## sort out missing value
dat <- mv.zene(dat)

## summary of an individual column
vec.summ(dat[,2])
vec.summ.1(dat[,2])

## summary of data frame
summ   <- df.summ(dat)                        ## default: vec.summ
summ.1 <- df.summ(dat, method=vec.summ.1)

## summary by groups
by(dat, list(cls=cls), df.summ)

## User-defined summary function:
vec.segment <- function(x, bar=c("SD", "SE", "CI"))
{
  bar <- match.arg(bar)

  centre <- mean(x, na.rm = TRUE)

  if (bar == "SD") {
    stderr <- sd(x, na.rm = TRUE)        ## Standard derivation (SD)
    lower  <- centre - stderr
    upper  <- centre + stderr
  } else if (bar == "SE") {      ## Standard error(SE) of mean
    stderr <- sd(x, na.rm = TRUE)/sqrt(sum(!is.na(x)))
    ## stderr <- sqrt(var(x, na.rm = TRUE)/length(x[complete.cases(x)]))
    lower  <- centre - stderr
    upper  <- centre + stderr
  } else if (bar == "CI") {      ## Confidence interval (CI), here 95%.
    conf   <- t.test(x)$conf.int
    lower  <- conf[1]
    upper  <- conf[2]
  } else {
    stop("'method' invalid")
  }

  res <- c(lower=lower, centre=centre, upper=upper)
  return(res)
}

## test it
vec.segment(dat[,2])
summ.2 <- df.summ(dat, method=vec.segment, bar="SE")

## ------------------------------------------------------------
```

```
#' iris data
df.summ(iris)

#' Group summary
## library(plyr)
## ddply(iris, .(Species), df.summ)
## (tmp <- dlply(iris, .(Species), df.summ, method=vec.segment))
##do.call("rbind", tmp)

#' or you can use summarise to get the group summary for single variable:
## ddply(iris, .(Species), summarise,
##      mean=mean(Sepal.Length), std=sd(Sepal.Length))
```

`feat.agg` *Rank aggregation by Borda count algorithm*

Description

Use Borda count to get the final feature order.

Usage

```
feat.agg(fs.rank.list)
```

Arguments

`fs.rank.list`	A data frame of feature orders by different feature selectors.

Value

A list with components:

`fs.order`	Final feature order.
`fs.rank`	Aggregated rank list by Borda count.

Author(s)

Wanchang Lin

See Also

`feat.rank.re`, `feat.mfs`

Examples

```
data(abr1)
dat <- preproc(abr1$pos[,200:400], method="log10")
cls <- factor(abr1$fact$class)

## feature selection without resampling
fs <- feat.mfs(dat, cls, method=c("fs.anova","fs.rf","fs.rfe"),
               is.resam=FALSE)
## rank aggregation
fs.1 <- feat.agg(fs$fs.rank)
names(fs.1)
```

`feat.freq` *Frequency and Stability of Feature Selection*

Description

Frequency and stability of feature selection.

Usage

```
feat.freq(x,rank.cutoff=50,freq.cutoff=0.5)
```

Arguments

`x`	A matrix or data frame of feature orders.
`rank.cutoff`	A numeric value for cutoff of top features.
`freq.cutoff`	A numeric value for cutoff of feature frequency.

Value

A list with components:

`freq.all`	Feature frequencies.
`freq`	Feature frequencies larger than `freq.cutoff`.
`stability`	Stability rate of feature ranking.
`rank.cutoff`	Top feature order cut-off used.
`freq.cutoff`	Feature frequency cut-off used.

Author(s)

Wanchang Lin

References

Davis, C. A., et al., (2006) Reliable gene signatures for microarray classification: assessment of stability and performance. *Bioinformatics*, vol.22, no.19, 2356 - 2363.

Michiels, S., et al., (2005) Prediction of cancer outcome with microarrays: a multiple random validation strategy. *Lancet*, vol.365, 488 - 492.

See Also

feat.rank.re

Examples

```
## prepare data set
data(abr1)
cls <- factor(abr1$fact$class)
dat <- abr1$pos
## dat <- abr1$pos[,110:1930]

## fill zeros with NAs
dat <- mv.zene(dat)

## missing values summary
mv <- mv.stats(dat, grp=cls)
mv    ## View the missing value pattern

## filter missing value variables
## dim(dat)
dat <- dat[,mv$mv.var < 0.15]
## dim(dat)

## fill NAs with mean
dat <- mv.fill(dat,method="mean")

## log transformation
dat <- preproc(dat, method="log10")

## select class "1" and "2" for feature ranking
ind <- grepl("1|2", cls)
mat <- dat[ind,,drop=FALSE]
mat <- as.matrix(mat)
grp <- cls[ind, drop=TRUE]

## use resampling method of bootstrap
pars   <- valipars(sampling="boot",niter=10, nreps=5)
z      <- feat.rank.re(mat,grp,method="fs.plsvip",pars = pars)

## compute the frequency and stability of feature selection
freq <- feat.freq(z$order.list,rank.cutoff=50,freq.cutoff=0.5)
```

`feat.mfs` *Multiple Feature Selection*

Description

Multiple feature selection with or without resampling procedures.

Usage

```
feat.mfs(x,y,method,pars = valipars(),is.resam = TRUE, ...)

feat.mfs.stab(fs.res,rank.cutoff = 20,freq.cutoff = 0.5)

feat.mfs.stats(fs.stats,cumu.plot=FALSE, main="Stats Plot",
               ylab="Values", xlab="Index of variable", ...)
```

Arguments

`x`	A matrix or data frame containing the explanatory variables.
`y`	A factor specifying the class for each observation.
`method`	Multiple feature selection/ranking method to be used.
`pars`	A list of resampling scheme. See `valipars` for details.
`is.resam`	A logical value indicating whether the resampling should be applied.
`fs.res`	A list obtained by running `feat.mfs` .
`rank.cutoff`	Cutoff of top features for frequency calculating.
`freq.cutoff`	Cutoff of feature frequency.
`fs.stats`	A matrix of feature statistics or values outputted by `feat.mfs`
`cumu.plot`	A logical value indicating the cumulative scores should be plotted.
`main,xlab,ylab`	Plot parameters
`...`	Additional parameters.

Details

`feat.mfs.stab` summarises multiple feature selection only when resampling strategy is employed (i.e. `is.resam` is `TRUE` when calling `feat.mfs`). It obtains these results based on `feat.mfs`'s returned value called `all`.

`feat.mfs.stats` handles the statistical values or scores. Its purpose is to provide a guidance in selecting the best number of features by spotting the elbow point. This method should work in conjunction with plotting of p-values and their corresponding adjusted values such as FDR and Bonferroni in the multiple hypothesis test.

Value

`feat.mfs` returns a list with components:

`fs.order`	A data frame of feature order from best to worst.
`fs.rank`	A matrix of feature ranking scores.
`fs.stats`	A matrix of feature statistics or values.
`all`	A list of output of `feat.rank.re` for each feature selection method.

`feat.mfs.stab` returns a list with components:

`fs.freq`	Feature frequencies larger than `freq.cutoff`.
`fs.subs`	Feature with frequencies larger than `freq.cutoff`.
`fs.stab`	Stability rate of feature ranking.
`fs.cons`	A matrix of feature consensus table based on feature frequency.

`feat.mfs.stats` returns a list with components:

`stats.tab`	A statistical values with their corresponding names.
`stats.long`	Long-format of statistical values for plotting.
`stats.p`	An object of class "trellis".

Note

The feature order can be computed directly from the overall statistics `fs.stats`. It is, however, slightly different from `fs.order` obtained by rank aggregation when resampling is employed.

The `fs.cons` and `fs.freq` are computed based on `fs.order`.

Author(s)

Wanchang Lin

See Also

`feat.rank.re`, `feat.freq`

Examples

```
## Not run:
library(lattice)
data(abr1)
dat <- preproc(abr1$pos[,200:400], method="log10")
cls <- factor(abr1$fact$class)

tmp <- dat.sel(dat, cls, choices=c("1","2"))
x   <- tmp[[1]]$dat
y   <- tmp[[1]]$cls

fs.method <- c("fs.anova","fs.rf","fs.rfe")
```

```
fs.pars   <- valipars(sampling="cv",niter=10,nreps=5)
fs <- feat.mfs(x, y, fs.method, fs.pars)   ## with resampling
names(fs)

## frequency, consensus and stabilities of feature selection
fs.stab <- feat.mfs.stab(fs)
print(fs.stab$fs.cons,digits=2,na.print="")

## plot feature selection frequency
freq <- fs.stab$fs.freq
dotplot(freq$fs.anova, type="o", main="Feature Selection Frequencies")
barchart(freq$fs.anova)

## rank aggregation
fs.agg <- feat.agg(fs$fs.rank)

## stats table and plotting
fs.stats <- fs$fs.stats
tmp <- feat.mfs.stats(fs.stats, cumu.plot = TRUE)
tmp$stats.p
fs.tab <- tmp$stats.tab
## convert to matrix
fs.tab <- list2df(un.list(fs.tab))

## without resampling
fs.1 <- feat.mfs(x, y, method=fs.method, is.resam = FALSE)

## End(Not run)
```

`feat.rank.re` *Feature Ranking with Resampling Method*

Description

Feature selection with resampling method.

Usage

```
feat.rank.re(x,y,method=,pars = valipars(),tr.idx=NULL,...)
```

Arguments

`x`	A matrix or data frame containing the explanatory variables.
`y`	A factor specifying the class for each observation.
`method`	Feature selection method to be used. For each method used in this function, the output must be a list including two components, `fs.rank` (rank scores of features) and `fs.order`(feature orders in descending order).

pars	A list of resampling scheme method such as *Leave-one-out cross-validation*, *Cross-validation*, *Bootstrap* and *Randomised validation (holdout)*. See valipars for details.
tr.idx	User defined index of training samples. Can be generated by trainind.
...	Additional parameters to method.

Value

A list with components:

method	Feature selection method used.
fs.rank	A vector of final feature ranking list.
fs.order	A vector of final feature order from best to worst.
rank.list	Feature rank lists of all computation.
order.list	Feature order lists of all computation.
pars	Resampling parameters.
tr.idx	Index of training samples.
all	All results come from re-sampling.

Author(s)

Wanchang Lin

See Also

valipars, feat.freq, frankvali

Examples

```
## prepare data set
data(abr1)
cls <- factor(abr1$fact$class)
dat <- abr1$pos
## dat <- abr1$pos[,110:1930]

## fill zeros with NAs
dat <- mv.zene(dat)

## missing values summary
mv <- mv.stats(dat, grp=cls)
## mv    ## View the missing value pattern

## filter missing value variables
## dim(dat)
dat <- dat[,mv$mv.var < 0.15]
## dim(dat)

## fill NAs with mean
dat <- mv.fill(dat,method="mean")
```

```
## log transformation
dat <- preproc(dat, method="log10")

## select class "1" and "2" for feature ranking
ind <- which(cls==1 | cls==2)
x   <- dat[ind,,drop=FALSE]
y   <- cls[ind, drop=TRUE]

## feature selection
pars   <- valipars(sampling="boot",niter=2,nreps=5)
tr.idx <- trainind(y,pars=pars)

z      <- feat.rank.re(x,y,method="fs.auc",pars = pars)
names(z)
```

`frank.err` *Feature Ranking and Validation on Feature Subset*

Description

Get feature ranking on the training data and validate selected feature subsets by estimating their classification error rate.

Usage

```
frank.err(dat.tr, cl.tr, dat.te, cl.te, cl.method="svm",
          fs.method="fs.auc", fs.order=NULL, fs.len="power2", ...)
```

Arguments

`dat.tr`	A data frame or matrix of training data. Feature ranking and classification model are carried on this data set.
`cl.tr`	A factor or vector of training class.
`dat.te`	A data frame or matrix of test data. Error rates are calculated on this data set.
`cl.te`	A factor or vector of test class.
`cl.method`	Classification method to be used. Any classification methods can be employed if they have method `predict` (except `knn`) with output of predicted class label or one component with name of `class` in the returned list, such as `randomForest`, `svm`, `knn` and `lda`.
`fs.method`	Feature ranking method. If `fs.order` is not `NULL`, it is ignored.
`fs.order`	A vector of feature order. Default is `NULL` and then the feature selection will be performed on the training data.
`fs.len`	The lengths of feature subsets used for validation. For details, see `get.fs.len`.
`...`	Additional parameters to `fs.method` or `cl.method`.

Value

A list with components:

cl.method	Classification method used.
fs.len	The lengths of feature subsets used for validation.
error	Error rate for each feature length.
fs.method	Feature ranking method used.
fs.order	Feature order vector.
fs.rank	Feature ranking score vector.

Author(s)

Wanchang Lin

See Also

frankvali, get.fs.len

Examples

```
data(abr1)
dat <- abr1$pos
x   <- preproc(dat[,110:500], method="log10")
y   <- factor(abr1$fact$class)

dat <- dat.sel(x, y, choices=c("1","6"))
x.1 <- dat[[1]]$dat
y.1 <- dat[[1]]$cls

idx <- sample(1:nrow(x.1), round((2/3)*nrow(x.1)), replace=FALSE)
## construct train and test data
train.dat  <- x.1[idx,]
train.cl   <- y.1[idx]
test.dat   <- x.1[-idx,]
test.cl    <- y.1[-idx]

## validate feature selection on some feature subsets
res <- frank.err(train.dat, train.cl, test.dat, test.cl,
                 cl.method="knn", fs.method="fs.auc",
                 fs.len="power2")
names(res)
## full feature order list
res$fs.order

## validation on subsets of feature order
res$error

## or first apply feature selection
fs <- fs.auc(train.dat,train.cl)
## then apply error estimation for each selected feature subset
```

```
res.1 <- frank.err(train.dat, train.cl, test.dat, test.cl,
                   cl.method="knn", fs.order=fs$fs.order,
                   fs.len="power2")

res.1$error
```

---

frankvali *Estimates Feature Ranking Error Rate with Resampling*

Description

Estimates error rate of feature ranking with resampling methods.

Usage

```
frankvali(dat, ...)
## Default S3 method:
frankvali(dat,cl,cl.method = "svm", fs.method="fs.auc",
          fs.order=NULL, fs.len="power2", pars = valipars(),
          tr.idx=NULL,...)

## S3 method for class 'formula'
frankvali(formula, data = NULL, ..., subset, na.action = na.omit)

fs.cl(dat,cl,fs.order=colnames(dat), fs.len=1:ncol(dat),
      cl.method = "svm", pars = valipars(), all.fs=FALSE, ...)

fs.cl.1(dat,cl,fs.order=colnames(dat), cl.method = "svm",
        pars = valipars(), agg_f=FALSE,...)
```

Arguments

formula	A formula of the form groups ~ x1 + x2 + ... 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 formula are preferentially to be taken.
dat	A matrix or data frame containing the explanatory variables if no formula is given as the principal argument.
cl	A factor specifying the class for each observation if no formula principal argument is given.
cl.method	Classification method to be used. Any classification methods can be employed if they have method predict (except knn) with output of predicted class label or one component with name of class in the returned list, such as randomForest, svm, knn and lda.

fs.method	Feature ranking method to be used. If fs.order is not NULL, it will be overridden.
fs.order	A vector of ordered feature order. In frankvali its default is NULL and then the feature selection will be performed on the training data.
fs.len	Feature length used for validation. For details, see get.fs.len.
pars	A list of resampling scheme method such as *Cross-validation*, *Stratified cross-validation*, *Leave-one-out cross-validation*, *Randomised validation (holdout)*, *Bootstrap*, *.632 bootstrap* and *.632 plus bootstrap*, and control parameters for the calculation of accuracy. See valipars for details.
tr.idx	User defined index of training samples. Can be generated by trainind.
all.fs	A logical value indicating whether all features should be used for evaluation.
agg_f	A logical value indicating whether aggregated features should be used for evaluation.
...	Additional parameters to fs.method or cl.method.
subset	Optional vector, specifying a subset of observations to be used.
na.action	Function which indicates what should happen when the data contains NA's, defaults to na.omit.

Details

These functions validate the selected feature subsets by classification and resampling methods.

It can take any classification model if its argument format is model(formula, data, subset, ...) and their corresponding method predict.model(object, newdata, ...) can either return the only predicted class label or in a list with name as class, such as lda and pcalda.

The resampling method can be one of cv, scv, loocv, boot, 632b and 632pb.

The feature ranking method can take one of fs.rf, fs.auc, fs.welch, fs.anova, fs.bw, fs.snr, fs.kruskal, fs.relief and fs.rfe.

Value

frankvali returns an object of class including the components:

fs.method	Feature ranking method used.
cl.method	Classification method used.
fs.len	Feature lengths used.
fs.rank	Final feature ranking. It is obtained based on fs.list by Borda vote method.
err.all	Error rate for all computation.
err.iter	Error rate for each iteration.
err.avg	Average error rate for all iterations.
sampling	Sampling scheme used.
niter	Number of iterations.
nboot	Number of bootstrap replications if the sampling method is one of boot, 632b and 632pb.

nfold	Fold of cross-validations if the sampling is cv or scv.
nrand	Number of replications if the sampling is random.
fs.list	Feature list of all computation if fs.order is NULL.

fs.cl and fs.cl.1 return a matrix with columns of acc (accuracy), auc(area under ROC curve) and mar(class margin).

Note

fs.cl is the simplified version of frankvali. Both frankvali and fs.cl are used for validation of aggregated features from top to bottom only, but fs.cl.1 can be used for validation of either individual or aggregated features.

Author(s)

Wanchang Lin

See Also

feat.rank.re, frank.err, valipars, boxplot.frankvali, get.fs.len

Examples

```
data(abr1)
dat <- abr1$pos
x   <- preproc(dat[,110:500], method="log10")
y   <- factor(abr1$fact$class)

dat <- dat.sel(x, y, choices=c("1","2"))
x.1 <- dat[[1]]$dat
y.1 <- dat[[1]]$cls

len  <- c(1:20,seq(25,50,5),seq(60,90,10),seq(100,300,50))
pars <- valipars(sampling="boot",niter=2, nreps=4)
res  <- frankvali(x.1,y.1,cl.method = "knn", fs.method="fs.auc",
                  fs.len=len, pars = pars)
res
summary(res)
boxplot(res)

## Not run:
## or apply feature selection with re-sampling procedure at first
fs  <- feat.rank.re(x.1,y.1,method="fs.auc",pars = pars)

## then estimate error of feature selection.
res.1  <- frankvali(x.1,y.1,cl.method = "knn", fs.order=fs$fs.order,
                    fs.len=len, pars = pars)
res.1

## use formula
data.bin <- data.frame(y.1,x.1)
```

```
pars <- valipars(sampling="cv",niter=2,nreps=4)
res.2  <- frankvali(y.1~., data=data.bin,fs.method="fs.rfe",fs.len=len,
                    cl.method = "knn",pars = pars)
res.2

## examples of fs.cl and fs.cl.1
fs <- fs.rf(x.1, y.1)
res.3 <- fs.cl(x.1,y.1,fs.order=fs$fs.order, fs.len=len,
               cl.method = "svm", pars = pars, all.fs=TRUE)

ord <- fs$fs.order[1:50]
## aggregated features
res.4 <- fs.cl.1(x.1,y.1,fs.order=ord, cl.method = "svm", pars = pars,
                 agg_f=TRUE)

## individual feature
res.5 <- fs.cl.1(x.1,y.1,fs.order=ord, cl.method = "svm", pars = pars,
                 agg_f=FALSE)


## End(Not run)
```

`fs.anova` *Feature Selection Using ANOVA*

Description

Feature selection using ANOVA.

Usage

```
fs.anova(x,y,...)
```

Arguments

`x`	A data frame or matrix of data set.
`y`	A factor or vector of class.
`...`	Arguments to pass to method.

Value

A list with components:

`fs.rank`	A vector of feature ranking scores.
`fs.order`	A vector of feature order from best to worst.
`stats`	A vector of statistics.
`pval`	A vector of p values.

Author(s)

Wanchang Lin

Examples

```
## prepare data set
data(abr1)
cls <- factor(abr1$fact$class)
dat <- abr1$pos
## dat <- abr1$pos[,110:1930]

## fill zeros with NAs
dat <- mv.zene(dat)

## missing values summary
mv <- mv.stats(dat, grp=cls)
mv    ## View the missing value pattern

## filter missing value variables
## dim(dat)
dat <- dat[,mv$mv.var < 0.15]
## dim(dat)

## fill NAs with mean
dat <- mv.fill(dat,method="mean")

## log transformation
dat <- preproc(dat, method="log10")

## select class "1" and "2" for feature ranking
ind <- grepl("1|2", cls)
mat <- dat[ind,,drop=FALSE]
mat <- as.matrix(mat)
grp <- cls[ind, drop=TRUE]

## apply ANOVA method for feature selection/ranking
res <- fs.anova(mat,grp)
names(res)
```

`fs.auc` *Feature Selection Using Area under Receiver Operating Curve (AUC)*

Description

Feature selection using area under receiver operating curve (AUC).

Usage

```
fs.auc(x,y,...)
```

Arguments

x	A data frame or matrix of data set.
y	A factor or vector of class.
...	Arguments to pass(current ignored).

Value

A list with components:

fs.rank	A vector of feature ranking scores.
fs.order	A vector of feature order from best to worst.
stats	A vector of measurements.

Note

This function is for two-class problem only.

Author(s)

Wanchang Lin

Examples

```
## prepare data set
data(abr1)
cls <- factor(abr1$fact$class)
dat <- abr1$pos
## dat <- abr1$pos[,110:1930]

## fill zeros with NAs
dat <- mv.zene(dat)

## missing values summary
mv <- mv.stats(dat, grp=cls)
mv    ## View the missing value pattern

## filter missing value variables
## dim(dat)
dat <- dat[,mv$mv.var < 0.15]
## dim(dat)

## fill NAs with mean
dat <- mv.fill(dat,method="mean")

## log transformation
dat <- preproc(dat, method="log10")

## select class "1" and "2" for feature ranking
ind <- grepl("1|2", cls)
mat <- dat[ind,,drop=FALSE]
```

```
mat <- as.matrix(mat)
grp <- cls[ind, drop=TRUE]

## apply AUC method for feature selection/ranking
res <- fs.auc(mat,grp)
names(res)
```

---

fs.bw *Feature Selection Using Between-Group to Within-Group (BW) Ratio*

Description

Feature selection using ratio of between-group to within-group sums of squares (BW).

Usage

```
fs.bw(x,y,...)
```

Arguments

x	A data frame or matrix of data set.
y	A factor or vector of class.
...	Arguments to pass(current ignored).

Value

A list with components:

fs.rank	A vector of feature ranking scores.
fs.order	A vector of feature order from best to worst.
stats	A vector of measurements.

Author(s)

Wanchang Lin

References

Dudoit, S., Fridlyand, J. and Speed, T.P. Comparison of discrimination methods for classification of tumours using gene expression data. *Journal of the American Statistical Association*. Vol.97, No.457, 77-87.

Examples

```
## prepare data set
data(abr1)
cls <- factor(abr1$fact$class)
dat <- abr1$pos
## dat <- abr1$pos[,110:1930]

## fill zeros with NAs
dat <- mv.zene(dat)

## missing values summary
mv <- mv.stats(dat, grp=cls)
mv    ## View the missing value pattern

## filter missing value variables
## dim(dat)
dat <- dat[,mv$mv.var < 0.15]
## dim(dat)

## fill NAs with mean
dat <- mv.fill(dat,method="mean")

## log transformation
dat <- preproc(dat, method="log10")

## select class "1" and "2" for feature ranking
ind <- grepl("1|2", cls)
mat <- dat[ind,,drop=FALSE]
mat <- as.matrix(mat)
grp <- cls[ind, drop=TRUE]

## apply BW ratio method for feature selection/ranking
res <- fs.bw(mat,grp)
names(res)
```

`fs.kruskal` *Feature Selection Using Kruskal-Wallis Test*

Description

Feature selection using Kruskal-Wallis test.

Usage

```
fs.kruskal(x,y,...)
```

Arguments

x	A data frame or matrix of data set.
y	A factor or vector of class.
...	Arguments to pass to method.

Value

A list with components:

fs.rank	A vector of feature ranking scores.
fs.order	A vector of feature order from best to worst.
stats	A vector of statistics.
pval	A vector of p values.

Author(s)

Wanchang Lin

Examples

```
## prepare data set
data(abr1)
cls <- factor(abr1$fact$class)
dat <- abr1$pos
## dat <- abr1$pos[,110:1930]

## fill zeros with NAs
dat <- mv.zene(dat)

## missing values summary
mv <- mv.stats(dat, grp=cls)
mv    ## View the missing value pattern

## filter missing value variables
## dim(dat)
dat <- dat[,mv$mv.var < 0.15]
## dim(dat)

## fill NAs with mean
dat <- mv.fill(dat,method="mean")

## log transformation
dat <- preproc(dat, method="log10")

## select class "1" and "2" for feature ranking
ind <- grepl("1|2", cls)
mat <- dat[ind,,drop=FALSE]
mat <- as.matrix(mat)
grp <- cls[ind, drop=TRUE]
```

```
## apply Kruskal-Wallis test method for feature selection/ranking
res <- fs.kruskal(mat,grp)
names(res)
```

`fs.pca` *Feature Selection by PCA*

Description

Feature selection using PCA loadings.

Usage

```
fs.pca(x,thres=0.8, ...)
```

Arguments

`x`	A data frame or matrix of data set.
`thres`	The threshold of the cumulative percentage of PC's explained variances.
`...`	Additional arguments to `prcomp`.

Details

Since PCA loadings is a matrix with respect to PCs, the Mahalanobis distance of loadings is applied to select the features. (Other ways, for example, the sum of absolute values of loadings, or squared root of loadings, can be used.)

It should be noticed that this feature selection method is unsupervised.

Value

A list with components:

`fs.rank`	A vector of feature ranking scores.
`fs.order`	A vector of feature order from best to worst.
`stats`	A vector of measurements.

Author(s)

Wanchang Lin

See Also

`feat.rank.re`

Examples

```
## prepare data set
data(abr1)
cls <- factor(abr1$fact$class)
dat <- abr1$pos
## dat <- abr1$pos[,110:1930]

## fill zeros with NAs
dat <- mv.zene(dat)

## missing values summary
mv <- mv.stats(dat, grp=cls)
mv    ## View the missing value pattern

## filter missing value variables
## dim(dat)
dat <- dat[,mv$mv.var < 0.15]
## dim(dat)

## fill NAs with mean
dat <- mv.fill(dat,method="mean")

## log transformation
dat <- preproc(dat, method="log10")

## select class "1" and "2" for feature ranking
ind <- grepl("1|2", cls)
mat <- dat[ind,,drop=FALSE]
mat <- as.matrix(mat)
grp <- cls[ind, drop=TRUE]

## feature selection by PCA
res <- fs.pca(dat)
names(res)
```

`fs.pls` *Feature Selection Using PLS*

Description

Feature selection using coefficient of regression and VIP values of PLS.

Usage

```
fs.pls(x,y, pls="simpls",ncomp=10,...)
fs.plsvip(x,y, ncomp=10,...)
fs.plsvip.1(x,y, ncomp=10,...)
fs.plsvip.2(x,y, ncomp=10,...)
```

Arguments

x	A data frame or matrix of data set.
y	A factor or vector of class.
pls	A method for calculating PLS scores and loadings. The following methods are supported: • simpls: SIMPLS algorithm. • kernelpls: kernel algorithm. • oscorespls: orthogonal scores algorithm. For details, see simpls.fit, kernelpls.fit and oscorespls.fit in package **pls**.
ncomp	The number of components to be used.
...	Arguments passed to or from other methods.

Details

fs.pls ranks the features by regression coefficient of PLS. Since the coefficient is a matrix due to the dummy multiple response variables designed for the classification (category) problem, the Mahalanobis distance of coefficient is applied to select the features. (Other ways, for example, the sum of absolute values of coefficient, or squared root of coefficient, can be used.)

fs.plsvip and fs.plsvip.1 carry out feature selection based on the the Mahalanobis distance and absolute values of PLS's VIP, respectively.

fs.plsvip.2 is similar to fs.plsvip and fs.plsvip.1, but the category response is not treated as dummy multiple response matrix.

Value

A list with components:

fs.rank	A vector of feature ranking scores.
fs.order	A vector of feature order from best to worst.
stats	A vector of measurements.

Author(s)

Wanchang Lin

See Also

feat.rank.re

Examples

```
## prepare data set
data(abr1)
cls <- factor(abr1$fact$class)
dat <- abr1$pos
## dat <- abr1$pos[,110:1930]

## fill zeros with NAs
dat <- mv.zene(dat)

## missing values summary
mv <- mv.stats(dat, grp=cls)
mv    ## View the missing value pattern

## filter missing value variables
## dim(dat)
dat <- dat[,mv$mv.var < 0.15]
## dim(dat)

## fill NAs with mean
dat <- mv.fill(dat,method="mean")

## log transformation
dat <- preproc(dat, method="log10")

## select class "1" and "2" for feature ranking
ind <- grepl("1|2", cls)
mat <- dat[ind,,drop=FALSE]
mat <- as.matrix(mat)
grp <- cls[ind, drop=TRUE]

## apply PLS methods for feature selection
res.pls      <- fs.pls(mat,grp, ncomp=4)
res.plsvip   <- fs.plsvip(mat,grp, ncomp=4)
res.plsvip.1 <- fs.plsvip.1(mat,grp, ncomp=4)
res.plsvip.2 <- fs.plsvip.2(mat,grp, ncomp=4)

## check differences among these methods
fs.order <- data.frame(pls      = res.pls$fs.order,
                       plsvip   = res.plsvip$fs.order,
                       plsvip.1 = res.plsvip.1$fs.order,
                       plsvip.2 = res.plsvip.2$fs.order)
head(fs.order, 20)
```

---

fs.relief *Feature Selection Using RELIEF Method*

Description

Feature selection using RELIEF method.

Usage

```
fs.relief(x,y, m=NULL, k=10, ...)
```

Arguments

x	A data frame or matrix of data set.
y	A factor or vector of class.
m	Number of instances to sample without replacement. Default is NULL which takes all instances for computation.
k	Number of nearest neighbours used to estimate feature relevance.
...	Arguments to pass to method (current ignore).

Details

This function implements the **Relief** algorithm's extension called **ReliefF**, which applies to multi-class problem and searches for k of its nearest neighbours from the same class, called *hits*, and also k nearest neighbours from each of the different classes, called *misses*.

Value

A list with components:

fs.rank	A vector of feature ranking scores.
fs.order	A vector of feature order from best to worst.
stats	A vector of measurements.

Author(s)

Wanchang Lin

References

Kira, K. and Rendel, L. (1992). The Feature Selection Problem: Traditional Methods and a new algorithm. *Proc. Tenth National Conference on Artificial Intelligence*, MIT Press, 129 - 134.

Kononenko, I., Simes, E., and Robnik-Sikonja, M. (1997). Overcoming the Myopia of Induction Learning Algorithms with RELIEFF. *Applied Intelligence*, Vol.7, 1, 39-55.

Kononenko, I. (1994) Estimating Attributes: Analysis and Extensions of RELIEF, *European Conference on Machine Learning*, Ed. Francesco Bergadano and Luc De Raedt, 171-182, Springer

Robnik-Sikonja, M. and Kononenko, I. (2003) Theoretical and Empirical Analysis of ReliefF and RReliefF, *Machine Learning*, 53, 23 - 69.

Examples

```
data(iris)
x <- subset(iris, select = -Species)
y <- iris$Species

fs <- fs.relief(x, y, m=20,k=10)
```

`fs.rf` *Feature Selection Using Random Forests (RF)*

Description

Feature selection using Random Forests (RF).

Usage

```
fs.rf(x,y,...)
fs.rf.1(x,y,fs.len="power2",...)
```

Arguments

`x`	A data frame or matrix of data set.
`y`	A factor or vector of class.
`fs.len`	Method or numeric sequence for feature lengths. For details, see `get.fs.len`
`...`	Arguments to pass to `randomForests`.

Details

`fs.rf.1` select features based on successively eliminating the least important variables.

Value

A list with components:

`fs.rank`	A vector of feature ranking scores.
`fs.order`	A vector of feature order from best to worst.
`stats`	A vector of measurements. For `fs.rf`, it is Random Forest important score. For `fs.rf.1`, it is a dummy variable (current ignored).

Author(s)

Wanchang Lin

Examples

```
data(abr1)
cls <- factor(abr1$fact$class)
dat <- abr1$pos

## fill zeros with NAs
dat <- mv.zene(dat)

## missing values summary
mv <- mv.stats(dat, grp=cls)
mv    ## View the missing value pattern

## filter missing value variables
dat <- dat[,mv$mv.var < 0.15]

## fill NAs with mean
dat <- mv.fill(dat,method="mean")

## log transformation
dat <- preproc(dat, method="log10")

## select class "1" and "2" for feature ranking
ind <- grepl("1|2", cls)
mat <- dat[ind,,drop=FALSE]
mat <- as.matrix(mat)
grp <- cls[ind, drop=TRUE]

## apply random forests for feature selection/ranking
res   <- fs.rf(mat,grp)
res.1 <- fs.rf.1(mat,grp)

## compare the results
fs <- cbind(fs.rf=res$fs.order, fs.rf.1=res.1$fs.order)

## plot the important score of 'fs.rf' (not 'fs.rf.1')
score <- res$stats
score <- sort(score, decreasing = TRUE)
plot(score)
```

---

fs.rfe *Feature Selection Using SVM-RFE*

Description

Feature selection using Support Vector Machine based on Recursive Feature Elimination (SVM-RFE)

Usage

```
fs.rfe(x,y,fs.len="power2",...)
```

Arguments

x	A data frame or matrix of data set.
y	A factor or vector of class.
fs.len	Method for feature lengths used in SVM-RFE computation. For details, see get.fs.len.
...	Arguments to pass to svm.

Value

A list with components:

fs.rank	A vector of feature ranking scroes.
fs.order	A vector of feature order from best to worst.

Author(s)

Wanchang Lin

See Also

feat.rank.re, get.fs.len

Examples

```
## prepare data set
data(abr1)
cls <- factor(abr1$fact$class)
dat <- abr1$pos
## dat <- abr1$pos[,110:1930]

## fill zeros with NAs
dat <- mv.zene(dat)

## missing values summary
mv <- mv.stats(dat, grp=cls)
mv    ## View the missing value pattern

## filter missing value variables
## dim(dat)
dat <- dat[,mv$mv.var < 0.15]
## dim(dat)

## fill NAs with mean
dat <- mv.fill(dat,method="mean")

## log transformation
dat <- preproc(dat, method="log10")

## select class "1" and "2" for feature ranking
ind <- grepl("1|2", cls)
```

```
mat <- dat[ind,,drop=FALSE]
mat <- as.matrix(mat)
grp <- cls[ind, drop=TRUE]

## apply RFE method for feature selection/ranking
res <- fs.rfe(mat,grp)
names(res)
```

---

fs.snr *Feature Selection Using Signal-to-Noise Ratio (SNR)*

Description

Feature selection using signal-to-noise ratio (SNR).

Usage

```
fs.snr(x,y,...)
```

Arguments

x	A data frame or matrix of data set.
y	A factor or vector of class.
...	Arguments to pass(current ignored).

Value

A list with components:

fs.rank	A vector of feature ranking scores.
fs.order	A vector of feature order from best to worst.
stats	A vector of measurements.

Note

This function is for two-class problem only.

Author(s)

Wanchang Lin

Examples

```
## prepare data set
data(abr1)
cls <- factor(abr1$fact$class)
dat <- abr1$pos
## dat <- abr1$pos[,110:1930]

## fill zeros with NAs
dat <- mv.zene(dat)

## missing values summary
mv <- mv.stats(dat, grp=cls)
mv    ## View the missing value pattern

## filter missing value variables
## dim(dat)
dat <- dat[,mv$mv.var < 0.15]
## dim(dat)

## fill NAs with mean
dat <- mv.fill(dat,method="mean")

## log transformation
dat <- preproc(dat, method="log10")

## select class "1" and "2" for feature ranking
ind <- grepl("1|2", cls)
mat <- dat[ind,,drop=FALSE]
mat <- as.matrix(mat)
grp <- cls[ind, drop=TRUE]

## apply SNR method for feature selection/ranking
res <- fs.snr(mat,grp)
names(res)
```

`fs.welch` *Feature Selection Using Welch Test*

Description

Feature selection using Welch test.

Usage

```
fs.welch(x,y,...)
```

Arguments

x	A data frame or matrix of data set.
y	A factor or vector of class.
...	Arguments to pass to method.

Value

A list with components:

fs.rank	A vector of feature ranking scores.
fs.order	A vector of feature order from best to worst.
stats	A vector of statistics.
pval	A vector of p values.

Note

This function is for two-class problem only.

Author(s)

Wanchang Lin

Examples

```
## prepare data set
data(abr1)
cls <- factor(abr1$fact$class)
dat <- abr1$pos
## dat <- abr1$pos[,110:1930]

## fill zeros with NAs
dat <- mv.zene(dat)

## missing values summary
mv <- mv.stats(dat, grp=cls)
mv    ## View the missing value pattern

## filter missing value variables
## dim(dat)
dat <- dat[,mv$mv.var < 0.15]
## dim(dat)

## fill NAs with mean
dat <- mv.fill(dat,method="mean")

## log transformation
dat <- preproc(dat, method="log10")

## select class "1" and "2" for feature ranking
ind <- grepl("1|2", cls)
```

```
mat <- dat[ind,,drop=FALSE]
mat <- as.matrix(mat)
grp <- cls[ind, drop=TRUE]

## apply Welch method for feature selection/ranking
res <- fs.welch(mat,grp)
names(res)
```

`fs.wilcox` *Feature Selection Using Wilcoxon Test*

Description

Feature selection using Wilcoxon test.

Usage

```
fs.wilcox(x,y,...)
```

Arguments

`x`	A data frame or matrix of data set.
`y`	A factor or vector of class.
`...`	Arguments to pass to method.

Value

A list with components:

`fs.rank`	A vector of feature ranking scores.
`fs.order`	A vector of feature order from best to worst.
`stats`	A vector of statistics.
`pval`	A vector of p values.

Note

This function is for two-class problem only.

Author(s)

Wanchang Lin

Examples

```
## prepare data set
data(abr1)
cls <- factor(abr1$fact$class)
dat <- abr1$pos
## dat <- abr1$pos[,110:1930]

## fill zeros with NAs
dat <- mv.zene(dat)

## missing values summary
mv <- mv.stats(dat, grp=cls)
mv    ## View the missing value pattern

## filter missing value variables
## dim(dat)
dat <- dat[,mv$mv.var < 0.15]
## dim(dat)

## fill NAs with mean
dat <- mv.fill(dat,method="mean")

## log transformation
dat <- preproc(dat, method="log10")

## select class "1" and "2" for feature ranking
ind <- grepl("1|2", cls)
mat <- dat[ind,,drop=FALSE]
mat <- as.matrix(mat)
grp <- cls[ind, drop=TRUE]

## apply Welch method for feature selection/ranking
res <- fs.wilcox(mat,grp)
names(res)
```

`get.fs.len` *Get Length of Feature Subset for Validation*

Description

Get feature lengths for feature selection validation by classification.

Usage

```
get.fs.len(p,fs.len=c("power2"))
```

Arguments

p	Number of features in the data set.
fs.len	Method or numeric sequence for feature lengths. It can be a numeric vector as user-defined feature lengths, or methods: • full. The feature lengths are p, ..., 2, 1. This is an exhaustive method. If p is too large, it will consume a lot of time and hence it is not practical. • half. The feature lengths are the sequence of p, p/2, p/2/2, ..., 1. • power2. The feature lengths are the sequence of p, 2^(log2(p)-1), ..., 2^1, 2^0.

Details

The generation of feature length is used in the validation of feature subsets by classification. The feature length decide the lengths of feature subset starting from top of the full feature order list.

Value

An descending order numeric vector of feature lengths.

Note

The last length of feature returned is always p.

Author(s)

Wanchang Lin

See Also

fs.rfe, frank.err, frankvali

Examples

```
data(abr1)
dat <- abr1$pos

## number of featres
p <- ncol(dat)

## predefined feature lengths. The returned will be descending order
## vector with the first one is p.
(vec <- get.fs.len(p, fs.len=c(1,2,3,4,5,6,7,8,9,10)))

## use all features as feature lengths
(vec.full <- get.fs.len(p, fs.len="full"))

## use "half"
(vec.half <- get.fs.len(p, fs.len="half"))
```

```
## use "power2"
(vec.power2 <- get.fs.len(p, fs.len="power2"))
```

---

grpplot *Plot Matrix-Like Object by Group*

Description

Plot matrix-like object by group

Usage

```
grpplot(x, y, plot = "pairs", ...)
```

Arguments

x	A matrix or data frame to be plotted.
y	A factor or vector giving group information of columns of x.
plot	One of plot types: strip, box, density and pairs.
...	Further arguments. See corresponding entry in xyplot for non-trivial details. One argument is ep: an integer for plotting ellipse. 1 and 2 for plotting overall and group ellipse, respectively. Otherwise, none. For details, see panel.elli.1.

Value

An object of class "trellis".

Author(s)

Wanchang Lin

See Also

panel.elli.1, pcaplot, pca_plot_wrap, lda_plot_wrap, pls_plot_wrap.

Examples

```
data(iris)
grpplot(iris[,1:4], iris[,5],plot="strip", main="IRIS DATA")
grpplot(iris[,1:4], iris[,5],plot="box", main="IRIS DATA")
grpplot(iris[,1:4], iris[,5],plot="density", main="IRIS DATA")
grpplot(iris[,1:4], iris[,5],plot="pairs",main="IRIS DATA",ep=2)

## returns an object of  class "trellis".
tmp <- grpplot(iris[,c(2,1)], iris[,5],main="IRIS DATA",ep=2)
tmp
```

```
## change symbol's color, type and size
grpplot(iris[,c(2,1)], iris[,5],main="IRIS DATA", cex=1.5,
       auto.key=list(space="right", col=c("black","blue","red")),
       par.settings = list(superpose.symbol = list(col=c("black","blue","red"),
                                                   pch=c(1:3))))
```

`list.util` *List Manipulation Utilities*

Description

Functions to handle manipulation of list.

Usage

```
list2df(x)

un.list(x, y="")

shrink.list(x)
```

Arguments

x	A list to be manipulated.
y	A character or string of separator.

Details

`list2df` converts a list with components of vector to a data frame. Shorter vectors will be filled with NA. It is useful to convert rugged vectors into a data frame which can be written to an Excel file.

`un.list` collapses higher-depths list to 1-depth list. This function uses recursive programming skill to tackle any depths of list.

`shrink.list` removes all NULL or NA entries from a list.

Value

`list2df` returns a data frame. `un.list` returns a list. `shrink.list` retuns a list.

Author(s)

Wanchang Lin

See Also

`feat.mfs`

Examples

```
 ## See examples of function feat.mfs for the usages of list2df and un.list.
a <- list(x=1, y=NA, z=NULL)
b <- list(x=1, y=NA)
c <- list(x=1, z=NULL)

shrink.list(a)
shrink.list(b)
shrink.list(c)
```

`maccest` *Estimation of Multiple Classification Accuracy*

Description

Estimation of classification accuracy by multiple classifiers with resampling procedure and comparisons of multiple classifiers.

Usage

```
maccest(dat, ...)
## Default S3 method:
maccest(dat, cl, method="svm", pars = valipars(),
        tr.idx = NULL, comp="anova",...)
## S3 method for class 'formula'
maccest(formula, data = NULL, ..., subset, na.action = na.omit)
```

Arguments

`formula`	A formula of the form `groups ~ x1 + x2 + ...` 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 `formula` are preferentially to be taken.
`dat`	A matrix or data frame containing the explanatory variables if no formula is given as the principal argument.
`cl`	A factor specifying the class for each observation if no formula principal argument is given.
`method`	A vector of multiple classification methods to be used. Classifiers, such as `randomForest`, `svm`, `knn` and `lda`, can be used. For details, see `note` below.
`pars`	A list of resampling scheme such as *Leave-one-out cross-validation*, *Cross-validation*, *Randomised validation (holdout)* and *Bootstrap*, and control parameters for the calculation of accuracy. See `valipars` for details.
`tr.idx`	User defined index of training samples. Can be generated by `trainind`.

comp	Comparison method of multiple classifier. If comp is anova, the multiple comparisons are performed by ANOVA and then the pairwise comparisons are performed by HSDTukey. If comp is fried, the multiple comparisons are performed by Friedman Test and the pairwise comparisons are performed by Wilcoxon Test.
...	Additional parameters to method.
subset	Optional vector, specifying a subset of observations to be used.
na.action	Function which indicates what should happen when the data contains NA's, defaults to na.omit.

Details

The accuracy rates for classification are obtained used techniques such as *Random Forest*, *Support Vector Machine*, *k-Nearest Neighbour Classification*, *Linear Discriminant Analysis* and *Linear Discriminant Analysis* based on sampling methods, including *Leave-one-out cross-validation*, *Cross-validation*, *Randomised validation (holdout)* and *Bootstrap*.

Value

An object of class maccest, including the components:

method	Classification method used.
acc	Accuracy rate.
acc.iter	Accuracy rate of each iteration.
acc.std	Standard derivation of accuracy rate.
mar	Prediction margin.
mar.iter	Prediction margin of each iteration.
auc	The area under receiver operating curve (AUC).
auc.iter	AUC of each iteration.
comp	Multiple comparison method used.
h.test	Hypothesis test results of multiple comparison.
gl.pval	Global or overall p-value.
mc.pval	Pairwise comparison p-values.
sampling	Sampling scheme used.
niter	Number of iteration.
nreps	Number of replications in each iteration.
conf.mat	Overall confusion matrix.
acc.boot	A list of bootrap error such as .632 and .632+ if the validation method is bootrap.

Note

The maccest can take any classification model if its argument format is model(formula, data, subset, na.action, ...) and their corresponding method predict.model(object, newdata, ...) can either return the only predicted class label or in a list with name as class, such as lda and pcalda.

As for the multiple comparisons by ANOVA, the following assumptions should be considered:

- The samples are randomly and independently selected.
- The populations are normally distributed.
- The populations all have the same variance.

All the comparisons are based on the results of all iteration.

aam.mcl is a simplified version which returns acc (accuracy), auc(area under ROC curve) and mar(class margin).

Author(s)

Wanchang Lin

See Also

accest, aam.mcl, valipars, plot.maccest trainind, boxplot.maccest,classifier

Examples

```
# Iris data
data(iris)
x      <- subset(iris, select = -Species)
y      <- iris$Species

method <- c("randomForest","svm","pcalda","knn")
pars   <- valipars(sampling="boot", niter = 3, nreps=5, strat=TRUE)
res    <- maccest(Species~., data = iris, method=method, pars = pars,
                  comp="anova")
## or
res    <- maccest(x, y, method=method, pars=pars, comp="anova")

res
summary(res)
plot(res)
boxplot(res)
oldpar <- par(mar = c(5,10,4,2) + 0.1)
plot(res$h.test$tukey,las=1)   ## plot the tukey results
par(oldpar)
```

mbinest *Binary Classification by Multiple Classifier*

Description

Binary classification by multiple classifier.

Usage

```
mbinest(dat, cl, choices = NULL, method, pars=valipars(),...)
```

Arguments

dat	A matrix or data frame containing the explanatory variables.
cl	A factor specifying the class for each observation.
choices	The vector or list of class labels to be chosen for binary classification. For details, see dat.sel.
method	Multiple classification methods to be used. For details, see maccest.
pars	A list of parameters of the resampling method. See valipars for details.
...	Additional parameters to method.

Value

A list with components:

all	All results of classification.
com	A matrix of the combinations of the binary class labels.
acc	A table of classification accuracy for the binary combination.
mar	Prediction margin.
auc	The area under receiver operating curve (AUC).
method	Classification methods used.
niter	Number of iterations.
sampling	Sampling scheme used.
nreps	Number of replications in each iteration if sampling is not loocv.

Author(s)

Wanchang Lin

See Also

maccest, maccest,valipars, dat.sel

Examples

```
## iris data set
data(iris)
dat    <- subset(iris, select = -Species)
cl     <- iris$Species
method <- c("svm","pcalda")

pars  <- valipars(sampling="cv",niter = 10, nreps = 5)
res   <- mbinest(dat,cl,choices=c("setosa"), method=method,
                  pars = pars, kernel="linear")

## combine prediction accuracy, AUC and margin
z      <- round(cbind(res$acc,res$auc,res$mar),digits=3)
colnames(z) <- c(paste(method,".acc", sep=""),paste(method,".auc", sep=""),
                 paste(method,".mar", sep=""))
```

`mc.anova` *Multiple Comparison by 'ANOVA' and Pairwise Comparison by 'HSDTukey Test'*

Description

Performs multiple comparison by `ANOVA` and pairwise comparison by `HSDTukey Test`.

Usage

```
mc.anova(x, ...)
```

Arguments

`x`	A matrix or data frame to be tested.
`...`	Additional arguments pass to `anova` or `HSDTukey test`.

Value

A list with components:

`anova`	Hypothesis test results of `anova`.
`tukey`	Hypothesis test results of `HSDTukey.test`.
`gl.pval`	Global or overall p value returned by `anova`.
`mc.pval`	Pairwise p value returned by `HSDTukey.test`.

Author(s)

Wanchang Lin

See Also

maccest, mc.fried

Examples

```
# Iris data
data(iris)
x      <- subset(iris, select = -Species)
y      <- iris$Species

method <- c("randomForest","svm","pcalda","knn")
pars   <- valipars(sampling="boot", niter = 10, nreps=4)
res    <- maccest(x, y, method=method, pars=pars, comp="anova")

res
htest <- mc.anova(res$acc.iter)

oldpar <- par(mar = c(5,10,4,2) + 0.1)
plot(htest$tukey,las=1)   ## plot the tukey results
par(oldpar)
```

---

mc.fried *Multiple Comparison by 'Friedman Test' and Pairwise Comparison by 'Wilcoxon Test'*

Description

Performs multiple comparison by Friedman test and pairwise comparison by Wilcoxon Test.

Usage

```
mc.fried(x, p.adjust.method = p.adjust.methods,...)
```

Arguments

x	A matrix or data frame to be tested.
p.adjust.method	Method for adjusting p values (see p.adjust).
...	Additional arguments pass to friedman.test or pairwise.wilcox.test.

Value

A list with components:

fried	Hypothesis test results of friedman.test.
wilcox	Hypothesis test results of pairwise.wilcox.test.
gl.pval	Global or overall p value returned by friedman.test.
mc.pval	Pairwise p value returned by pairwise.wilcox.test.

Author(s)

Wanchang Lin

See Also

maccest, mc.anova

Examples

```
# Iris data
data(iris)
x      <- subset(iris, select = -Species)
y      <- iris$Species

method <- c("randomForest","svm","pcalda","knn")
pars   <- valipars(sampling="cv", niter = 10, nreps=4)
res    <- maccest(x, y, method=method, pars=pars,
                  comp="fried",kernel="linear")

res

htest <- mc.fried(res$acc.iter)
```

---

mc.norm *Normality Test by Shapiro-Wilk Test*

Description

Perform Shapiro-Wilk normality test by `shapiro.test` and plot the density function and boxplot.

Usage

```
mc.norm(x, ...)
```

Arguments

x	A matrix or data frame to be tested.
...	Additional arguments pass to `shapiro.test`.

Value

Object of `shapiro.test`, boxplot and histogram.

Author(s)

Wanchang Lin

See Also

maccest, mc.anova

Examples

```
data(iris)
x      <- subset(iris, select = -Species)
y      <- iris$Species
method <- c("randomForest","svm","pcalda","knn")
pars   <- valipars(sampling="boot", niter = 10, nreps=10)
res    <- maccest(x, y, method=method, pars=pars,
                  comp="anova")
res
res$acc.iter
mc.norm(res$acc.iter)
```

---

mdsplot *Plot Classical Multidimensional Scaling*

Description

Plot metric MDS with categorical information.

Usage

```
mdsplot(x, y, method = "euclidean", dimen = c(1,2), ...)
```

Arguments

x	A matrix or data frame to be plotted.
y	A factor or vector giving group information of columns of x.
method	The distance measure to be used. This must be one of "euclidean", "maximum", "manhattan", "canberra", "binary" or "minkowski". Any unambiguous substring can be given. It is only for mds_plot_wrap.
dimen	A vector of index of dimentonal to be plotted. Only two dimentions are are allowed.
...	Further arguments to prcomp or lattice. See corresponding entry in xyplot for non-trivial details of lattice. For pcaplot, one argument is ep: an integer for plotting 95% ellipse. 1 and 2 for plotting overall and group ellipse, respectively. Otherwise, none. For details, see panel.elli.1.

Value

mdsplot returns an object of class "trellis".

Author(s)

Wanchang Lin

See Also

grpplot, panel.elli, mds_plot_wrap, pcaplot

Examples

```
## examples of 'mdsplot'

data(iris)
x <- iris[,1:4]
y <- iris[,5]
mdsplot(x,y, dimen=c(1,2),ep=2)
mdsplot(x,y, dimen=c(2,1),ep=1)

tmp <- mdsplot(x,y, ep=2, conf.level = 0.9)
tmp

## change symbol's color, type and size
mdsplot(x, y, main="IRIS DATA", cex=1.2,
  auto.key=list(space="right", col=c("black","blue","red"), cex=1.2),
  par.settings = list(superpose.symbol = list(col=c("black","blue","red"),
                                              pch=c(1:3))))
```

---

mv.util *Missing Value Utilities*

Description

Functions to handle missing values of data set.

Usage

```
mv.stats(dat,grp=NULL,...)

mv.fill(dat,method="mean",ze_ne = FALSE)

mv.zene(dat)
```

Arguments

dat	A data frame or matrix of data set.
grp	A factor or vector of class.
method	Univariate imputation method for missing value. For details, see examples below.
ze_ne	A logical value indicating whether the zeros or negatives should be treated as missing values.
...	Additional parameters to mv.stats for plotting using **lattice**.

Value

`mv.fill` returns an imputed data frame.

`mv.zene` returns an NA-filled data frame.

`mv.stats` returns a list including the components:

- `mv.overall`: Overall missng value rate.
- `mv.var`: Missing value rate per variable (column).
- `mv.grp`: A matrix of missing value rate for different groups if argument `grp` is given.
- `mv.grp.plot`: An object of class `trellis` for plotting of `mv.grp` if argument `grp` is given.

Author(s)

Wanchang Lin

Examples

```
data(abr1)
dat <- abr1$pos[,1970:1980]
cls <- factor(abr1$fact$class)

## fill zeros with NAs
dat <- mv.zene(dat)

## missing values summary
mv <- mv.stats(dat, grp=cls)
plot(mv$mv.grp.plot)

## fill NAs with mean
dat.mean <- mv.fill(dat,method="mean")

## fill NAs with median
dat.median <- mv.fill(dat,method="median")

## ------------------------------------------------------------------------
## fill NAs with user-defined methods: two examples given here.
## a.) Random imputation function:
rand <- function(x,...) sample(x[!is.na(x)], sum(is.na(x)), replace=TRUE)

## test this function:
(tmp <- dat[,1])        ## an vector with NAs
## get the randomised values for NAs
rand(tmp)

## fill NAs with method "rand"
dat.rand <- mv.fill(dat,method="rand")

## b.) "Low" imputation function:
"low" <- function(x, ...) {
  max(mean(x,...) - 3 * sd(x,...), min(x, ...)/2)
}
```

```
## fill NAs with method "low"
dat.low <- mv.fill(dat, method="low")

## summary of imputed data set
df.summ(dat.mean)
```

---

osc *Orthogonal Signal Correction (OSC)*

Description

Data pre-processing by orthogonal signal correction (OSC).

Usage

```
osc(x, ...)

## Default S3 method:
osc(x, y, method="wold",center=TRUE,osc.ncomp=4,pls.ncomp=10,
   tol=1e-3, iter=20,...)

## S3 method for class 'formula'
osc(formula, data = NULL, ..., subset, na.action = na.omit)
```

Arguments

formula	A formula of the form groups ~ x1 + x2 + ... 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 formula are preferentially to be taken.
x	A matrix or data frame containing the explanatory variables if no formula is given as the principal argument.
y	A factor specifying the class for each observation if no formula principal argument is given.
method	A method for calculating OSC weights, loadings and scores. The following methods are supported: • wold: Original Wold et al approach. • sjoblom: Sjoblom et al approach. • wise: Wise and Gallagher approach.
center	A logical value indicating whether the data set should be centred by column-wise.
osc.ncomp	The number of components to be used in the OSC calculation.
pls.ncomp	The number of components to be used in the PLS calculation.

tol	A scalar value of tolerance for OSC computation.
iter	The number of iteration used in OSC calculation.
...	Arguments passed to or from other methods.
subset	An index vector specifying the cases to be used in the training sample.
na.action	A function to specify the action to be taken if NAs 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.

Value

An object of class osc containing the following components:

x	A matrix of OSC corrected data set.
R2	R2 statistics. It is calculated as the fraction of variation in X after OSC correction for the calibration (training) data.
angle	An angle used for checking if scores t is orthogonal to y. An angle close to 90 degree means that orthogonality is achieved in the correction process.
w	A matrix of OSC weights.
p	A matrix of OSC loadings.
t	A matrix of OSC scores.
call	The (matched) function call.
center	A logical value indicating whether the data set has been centred by column-wise.
osc.ncomp	The number of component used in OSC computation.
pls.ncomp	The number of component used in PLS computation.
method	The OSC algorithm used.

Note

This function may be called giving either a formula and optional data frame, or a matrix and grouping factor as the first two arguments.

Author(s)

Wanchang Lin

References

Wold, S., Antti, H., Lindgren, F., Ohman, J.(1998). Orthogonal signal correction of near infrared spectra. *Chemometrics Intell. Lab. Syst.*, 44: 175-185.

Westerhuis, J. A., de Jong, S., Smilde, A, K. (2001). Direct orthogonal signal correction. *Chemometrics Intell. Lab. Syst.*, 56: 13-25.

Sjoblom. J., Svensson, O., Josefson, M., Kullberg, H., Wold, S. (1998). An evaluation of orthogonal signal correction applied to calibration transfer of near infrared spectra. *Chemometrics Intell. Lab. Syst.*,44: 229-244.

Svensson, O., Kourti, T. and MacGregor, J.F. (2002). An investigation of orthogonal correction algorithms and their characteristics. *Journal of Chemometrics*, 16:176-188.

Wise, B. M. and Gallagher, N.B. *http://www.eigenvector.com/MATLAB/OSC.html*.

See Also

predict.osc, osc_wold, osc_sjoblom, osc_wise

Examples

```
data(abr1)
cl   <- factor(abr1$fact$class)
dat  <- abr1$pos

## divide data as training and test data
idx <- sample(1:nrow(dat), round((2/3)*nrow(dat)), replace=FALSE)

## construct train and test data
train.dat  <- dat[idx,]
train.t    <- cl[idx]
test.dat   <- dat[-idx,]
test.t     <- cl[-idx]

## build OSC model based on the training data
res <- osc(train.dat, train.t, method="wise", osc.ncomp=2, pls.ncomp=4)
names(res)
res
summary(res)

## pre-process test data by OSC
test.dat.1 <- predict(res,test.dat)$x
```

---

osc_sjoblom *Orthogonal Signal Correction (OSC) Approach by Sjoblom et al.*

Description

Orthogonal signal correction (OSC) approach by Sjoblom et al.

Usage

```
osc_sjoblom(x, y, center=TRUE,osc.ncomp=4,pls.ncomp=10,
            tol=1e-3,iter=20,...)
```

Arguments

x	A numeric data frame or matrix to be pre-processed.
y	A vector or factor specifying the class for each observation.
center	A logical value indicating whether the data set should be centred by column-wise.
osc.ncomp	The number of components to be used in the OSC calculation.

pls.ncomp	The number of components to be used in the PLS calculation.
tol	A scalar value of tolerance for OSC computation.
iter	The number of iteration used in OSC calculation.
...	Arguments passed to or from other methods.

Value

A list containing the following components:

x	A matrix of OSC corrected data set.
R2	R2 statistics. It is calculated as the fraction of variation in X after OSC correction.
angle	An angle used for checking if scores t is orthogonal to y. An angle close to 90 degree means that orthogonality is achieved in the correction process.
w	A matrix of OSC weights.
p	A matrix of OSC loadings.
t	A matrix of OSC scores.
center	A logical value indicating whether the data set has been centred by column-wise.

Author(s)

Wanchang Lin

References

Sjoblom. J., Svensson, O., Josefson, M., Kullberg, H., Wold, S. (1998). An evaluation of orthogonal signal correction applied to calibration transfer of near infrared spectra. *Chemometrics Intell. Lab. Syst.*,44: 229-244.

Svensson, O., Kourti, T. and MacGregor, J.F. (2002). An investigation of orthogonal correction algorithms and their characteristics. *Journal of Chemometrics*, 16:176-188.

Westerhuis, J. A., de Jong, S., Smilde, A, K. (2001). Direct orthogonal signal correction. *Chemometrics Intell. Lab. Syst.*, 56: 13-25.

See Also

osc, predict.osc, osc_wold, osc_wise

Examples

```
data(abr1)
cl   <- factor(abr1$fact$class)
dat  <- abr1$pos

## divide data as training and test data
idx <- sample(1:nrow(dat), round((2/3)*nrow(dat)), replace=FALSE)

## construct train and test data
```

```
train.dat  <- dat[idx,]
train.t    <- cl[idx]
test.dat   <- dat[-idx,]
test.t     <- cl[-idx]

## build OSC model based on the training data
res <- osc_sjoblom(train.dat, train.t)
names(res)

## pre-process test data by OSC
test.dat.1 <- predict.osc(res,test.dat)$x
```

`osc_wise` *Orthogonal Signal Correction (OSC) Approach by Wise and Gallagher.*

Description

Orthogonal signal correction (OSC) approach by Wise and Gallagher.

Usage

```
osc_wise(x, y, center=TRUE,osc.ncomp=4,pls.ncomp=10,
         tol=1e-3,iter=20,...)
```

Arguments

`x`	A numeric data frame or matrix to be pre-processed.
`y`	A vector or factor specifying the class for each observation.
`center`	A logical value indicating whether the data set should be centred by column-wise.
`osc.ncomp`	The number of components to be used in the OSC calculation.
`pls.ncomp`	The number of components to be used in the PLS calculation.
`tol`	A scalar value of tolerance for OSC computation.
`iter`	The number of iteration used in OSC calculation.
`...`	Arguments passed to or from other methods.

Value

A list containing the following components:

`x`	A matrix of OSC corrected data set.
`R2`	R2 statistics. It is calculated as the fraction of variation in X after OSC correction.

angle	An angle used for checking if scores t is orthogonal to y. An angle close to 90 degree means that orthogonality is achieved in the correction process.
w	A matrix of OSC weights.
p	A matrix of OSC loadings.
t	A matrix of OSC scores.
center	A logical value indicating whether the data set has been centred by column-wise.

Author(s)

Wanchang Lin

References

Westerhuis, J. A., de Jong, S., Smilde, A, K. (2001). Direct orthogonal signal correction. *Chemometrics Intell. Lab. Syst.*, 56: 13-25.

Wise, B. M. and Gallagher, N.B. *http://www.eigenvector.com/MATLAB/OSC.html*.

See Also

osc, predict.osc, osc_sjoblom, osc_wold

Examples

```
data(abr1)
cl   <- factor(abr1$fact$class)
dat  <- abr1$pos

## divide data as training and test data
idx <- sample(1:nrow(dat), round((2/3)*nrow(dat)), replace=FALSE)

## construct train and test data
train.dat  <- dat[idx,]
train.t    <- cl[idx]
test.dat   <- dat[-idx,]
test.t     <- cl[-idx]

## build OSC model based on the training data
res <- osc_wise(train.dat, train.t)
names(res)

## pre-process test data by OSC
test.dat.1 <- predict.osc(res,test.dat)$x
```

`osc_wold` *Orthogonal Signal Correction (OSC) Approach by Wold et al.*

Description

Orthogonal signal correction (OSC) approach by Wold et al.

Usage

```
osc_wold(x, y, center=TRUE,osc.ncomp=4,pls.ncomp=10,
         tol=1e-3,iter=20,...)
```

Arguments

`x`	A numeric data frame or matrix to be pre-processed.
`y`	A vector or factor specifying the class for each observation.
`center`	A logical value indicating whether the data set should be centred by column-wise.
`osc.ncomp`	The number of components to be used in the OSC calculation.
`pls.ncomp`	The number of components to be used in the PLS calculation.
`tol`	A scalar value of tolerance for OSC computation.
`iter`	The number of iteration used in OSC calculation.
`...`	Arguments passed to or from other methods.

Value

A list containing the following components:

`x`	A matrix of OSC corrected data set.
`R2`	R2 statistics. It is calculated as the fraction of variation in X after OSC correction.
`angle`	An angle used for checking if scores `t` is orthogonal to `y`. An angle close to 90 degree means that orthogonality is achieved in the correction process.
`w`	A matrix of OSC weights.
`p`	A matrix of OSC loadings.
`t`	A matrix of OSC scores.
`center`	A logical value indicating whether the data set has been centred by column-wise.

Author(s)

Wanchang Lin

References

Wold, S., Antti, H., Lindgren, F., Ohman, J.(1998). Orthogonal signal correction of nearinfrared spectra. *Chemometrics Intell. Lab. Syst.*, 44: 175-185.

Svensson, O., Kourti, T. and MacGregor, J.F. (2002). An investigation of orthogonal correction algorithms and their characteristics. *Journal of Chemometrics*, 16:176-188.

Westerhuis, J. A., de Jong, S., Smilde, A, K. (2001). Direct orthogonal signal correction. *Chemometrics Intell. Lab. Syst.*, 56: 13-25.

See Also

osc, predict.osc, osc_sjoblom, osc_wise

Examples

```
data(abr1)
cl   <- factor(abr1$fact$class)
dat  <- abr1$pos

## divide data as training and test data
idx <- sample(1:nrow(dat), round((2/3)*nrow(dat)), replace=FALSE)

## construct train and test data
train.dat  <- dat[idx,]
train.t    <- cl[idx]
test.dat   <- dat[-idx,]
test.t     <- cl[-idx]

## build OSC model based on the training data
res <- osc_wold(train.dat, train.t)
names(res)

## pre-process test data by OSC
test.dat.1 <- predict.osc(res,test.dat)$x
```

---

panel.elli *Panel Function for Plotting Ellipse and outlier*

Description

lattice panel functions for plotting grouped ellipse and outlier

Usage

```
panel.elli(x, y, groups = NULL,conf.level = 0.975, ...)
panel.elli.1(x, y, subscripts, groups=NULL, conf.level = 0.975,
             ep=0, com.grp=NULL, no.grp=NULL, ell.grp=NULL, ...)
panel.outl(x, y, subscripts, groups=NULL, conf.level = 0.975, labs, ...)
```

Arguments

x, y	Variables to be plotted.
conf.level	Confident level for ellipse
groups, subscripts	Internal parameters for Lattice.
labs	Labels for potential outliers.
ep	An integer for plotting ellipse. 1 and 2 for plotting overall and group ellipse, respectively. Otherwise, none.
com.grp	A list of characters to select which combination of groups to be plotted.
no.grp	A list of characters to select which individual group not to be plotted. Note it will be overridden by com.grp. If no com.grp and no.grp, ellipses of each individual group will be plotted.
ell.grp	Another categorical vector used for plotting ellipse. If provided, ep,com.grp and no.grp will be ignored. It should be different from default groups, but has the same length of groups. For details, see example below.
...	Further arguments. See corresponding entry in xyplot for non-trivial details.

Details

panel.elli is modified from function panel.ellipse in package **latticeExtra**.

panel.elli.1 gives more control on how to plot ellipse for the current group. It also provides an option to plot ellipse based on another user-defined groups.

panel.outl plots the labels of data points outside the ellipse. These data points can be treated as potential outliers.

Value

Retuns objects of class "trellis".

Note

panel.elli.1 can be called by functions grpplot, pcaplot, mdsplot, pca_plot_wrap, mds_plot_wrap, pls_plot_wrap and lda_plot_wrap by passing argument of ep. See examples of these function for details.

Author(s)

Wanchang Lin

See Also

grpplot, pcaplot, mdsplot.

Examples

```
library(lattice)
data(iris)

## ======================================================================
## Examples of calling 'panel.elli' and 'panel.outl'
xyplot(Sepal.Length ~ Petal.Length, data = iris, groups=Species,
        par.settings = list(superpose.symbol = list(pch=c(15:17)),
                            superpose.line = list(lwd=2, lty=1:3)),
        panel = function(x, y, ...) {
            panel.xyplot(x, y, ...)
            panel.elli(x, y, ..., type="l",lwd=2)
            panel.outl(x,y, ...)
        },
        auto.key = list(x = .1, y = .8, corner = c(0, 0)),
        labs=rownames(iris), conf.level=0.9,adj = -0.5)

## Without groups
xyplot(Sepal.Length ~ Petal.Length, data = iris,
        par.settings = list(plot.symbol = list(cex = 1.1, pch=16)),
        panel = function(x, y, ...) {
            panel.xyplot(x, y, ...)
            panel.elli(x, y, ..., type="l", lwd = 2)
            panel.outl(x,y, ...)
        },
        auto.key = list(x = .1, y = .8, corner = c(0, 0)),
        labs=rownames(iris), conf.level=0.9,adj = -0.5)


## With conditioning
xyplot(Sepal.Length ~ Petal.Length|Species, data = iris,
        par.settings = list(plot.symbol = list(cex = 1.1, pch=16)),
        layout=c(2,2),
        panel = function(x, y, ...) {
            panel.xyplot(x, y, ...)
            panel.elli(x, y, ..., type="l", lwd = 2)
            panel.outl(x,y, ...)
        },
        auto.key = list(x = .6, y = .8, corner = c(0, 0)),
        adj = 0,labs=rownames(iris), conf.level=0.95)

## ======================================================================
## Examples of 'panel.elli.1'
xyplot(Sepal.Length ~ Petal.Length, data = iris, groups=Species,
        ## ------------------------------------------------
        ## Select what to be plotted.
        ep=2,
        ## com.grp = list(a="setosa",b=c("versicolor", "virginica")),
        ## no.grp = "setosa", ## Not draw ellipse for "setosa"
        ## ------------------------------------------------
        par.settings = list(superpose.symbol = list(pch=c(15:17)),
                            superpose.line = list(lwd=2, lty=1:3)),
```

```
        panel = function(x, y, ...) {
            panel.xyplot(x, y, ...)
            panel.elli.1(x, y, ...)
            panel.outl(x,y, ...)
        },
        auto.key = list(points = TRUE, rectangles = FALSE, space = "right"),
        adj = 0,labs=rownames(iris), conf.level=0.95)

xyplot(Sepal.Length ~ Petal.Length, data = iris, groups=Species,
        ## ----------------------------------------------------
        ## Select what to be plotted.
        ep=2,
        ## com.grp = list(a="setosa",b=c("versicolor", "virginica")),
        no.grp = c("setosa","versicolor"),## Only draw "virginica"
        ## ----------------------------------------------------
        par.settings = list(superpose.symbol = list(pch=c(15:17)),
                            superpose.line = list(lwd=2, lty=1:3)),
        panel = function(x, y, ...) {
            panel.xyplot(x, y, ...)
            panel.elli.1(x, y, ...)
        },
        auto.key = list(x = .1, y = .8, corner = c(0, 0)))

xyplot(Sepal.Length ~ Petal.Length, data = iris, groups=Species,
        ## ----------------------------------------------------
        ## Select what to be plotted.
        ep=2,
        com.grp = list(a="setosa",b=c("versicolor", "virginica")),
        ## no.grp = "setosa", ## Not draw ellipse for "setosa"
        ## ----------------------------------------------------
        par.settings = list(superpose.symbol = list(pch=c(15:17)),
                            superpose.line = list(lwd=2, lty=1:3)),
        panel = function(x, y, ...) {
            panel.xyplot(x, y, ...)
            panel.elli.1(x, y, ...)
        },
        auto.key = list(x = .1, y = .8, corner = c(0, 0)))

  xyplot(Sepal.Length ~ Petal.Length, data = iris, groups=Species, ep=1,
        par.settings = list(superpose.symbol = list(pch=c(15:17)),
                            superpose.line = list(lwd=2, lty=1:3)),
        panel = function(x, y, ...) {
            panel.xyplot(x, y, ...)
            panel.elli.1(x, y, ...)
        },
        auto.key = list(points = TRUE, rectangles = FALSE, space = "right"))

## ======================================================================
## Another data set from package MASS
require(MASS)
data(Cars93)

## Plot ellipse based on original groups: DriveTrain
```

```
xyplot(Price~EngineSize, data=Cars93, groups=DriveTrain, ep=2,
        par.settings = list(superpose.symbol = list(pch=c(15:17)),
                            superpose.line = list(lwd=2, lty=1:3)),
        panel = function(x, y, ...) {
            panel.xyplot(x, y, ...)
            panel.elli.1(x, y, ...)
        },
        auto.key = list(points = TRUE, rectangles = FALSE, space = "right"))

## But we want to plot ellipse using AirBags
xyplot(Price~EngineSize, data=Cars93, groups=DriveTrain,
        ell.grp=Cars93$AirBags,
        par.settings = list(superpose.symbol = list(pch=c(15:17)),
                            superpose.line = list(lwd=2, lty=1:3)),
        panel = function(x, y, ...) {
            panel.xyplot(x, y, ...)
            panel.elli.1(x, y, ...)
        },
        auto.key = list(points = TRUE, rectangles = FALSE, space = "right"))
```

`panel.smooth.line` *Panel Function for Plotting Regression Line*

Description

lattice panel function for plotting regression line with red colour.

Usage

```
panel.smooth.line(x, y,...)
```

Arguments

`x, y`	Variables to be plotted.
`...`	Further arguments. See corresponding entry in `xyplot` for non-trivial details.

Value

An object of class `"trellis"`.

Author(s)

Wanchang Lin

Examples

```
library(lattice)
data(iris)
splom(~iris[,1:4], varname.cex = 1.0, pscales = 0, panel = panel.smooth.line)
```

`pca.outlier` *Outlier detection by PCA*

Description

Outlier detection by the Mahalanobis distances of PC1 and PC2. Also plot PC1 and PC2 with its confidence ellipse.

Usage

```
pca.outlier(x, center = TRUE, scale=TRUE,conf.level = 0.975,...)

pca.outlier.1(x, center = TRUE, scale=TRUE, conf.level = 0.975,
            group=NULL, main = "PCA", cex=0.7,...)
```

Arguments

`x`	A data frame or matrix.
`center`	A logical value indicating whether the variables should be shifted to be zero centred before PCA analysis takes place.
`scale`	A logical value indicating whether the variables should be scaled to have unit variance before PCA analysis takes place.
`conf.level`	The confidence level for controlling the cutoff of the Mahalanobis distances.
`group`	A string character or factor indicating group information of row of `x`. It is used only for plotting.
`main`	An overall title for PCA plot.
`cex`	A numerical value giving the amount by which plotting text and symbols should be magnified relative to the default.
`...`	Further arguments for plotting

Value

A list with components:

`plot`	plot object of class `"trellis"` by `pca.outlier` only.
`outlier`	Outliers detected.
`conf.level`	Confidence level used.
`mah.dist`	Mahalanobis distances of each data sample.
`cutoff`	Cutoff of Mahalanobis distances used for outlier detection.

Note

Examples of `panel.elli` and `panel.outl` give more general information about ellipses and outliers. If you ONLY want to plot outliers based on PCA in a general way, for example, outliers in different groups or in conditional panel, you can write an wrapper function combining with `pca.comp`, `panel.elli` and `panel.outl`. It is quite similiar to the implementation of `pca_plot_wrap`.

Author(s)

Wanchang Lin

See Also

pcaplot, grpplot, panel.outl,panel.elli, pca_plot_wrap

Examples

```
data(iris)

## call lattice version
pca.outlier(iris[,1:4], adj=-0.5)
## plot group
pca.outlier(iris[,1:4], adj=-0.5,groups=iris[,5])
## more information about groups
pca.outlier(iris[,1:4],groups=iris[,5],adj = -0.5, xlim=c(-5, 5),
              auto.key = list(x = .05, y = .9, corner = c(0, 0)),
              par.settings = list(superpose.symbol=list(pch=rep(1:25))))

## call basic graphic version
pca.outlier.1(iris[,1:4])
## plot group
pca.outlier.1(iris[,1:4], group=iris[,5])
```

---

pcalda *Classification with PCADA*

Description

Classification with combination of principal component analysis (PCA) and linear discriminant analysis (LDA).

Usage

```
pcalda(x, ...)

## Default S3 method:
pcalda(x, y, center = TRUE, scale. = FALSE, ncomp = NULL,
       tune=FALSE,...)

## S3 method for class 'formula'
pcalda(formula, data = NULL, ..., subset, na.action = na.omit)
```

Arguments

formula	A formula of the form groups ~ x1 + x2 + ... 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 formula are preferentially to be taken.
x	A matrix or data frame containing the explanatory variables if no formula is given as the principal argument.
y	A factor specifying the class for each observation if no formula principal argument is given.
center	A logical value indicating whether x should be shifted to zero centred by column-wise.
scale.	A logical value indicating whether x should be scaled to have unit variance by column-wise before the analysis takes place.
ncomp	The number of principal components to be used in the classification. If NULL and tune=TRUE, it is the row number of x minus the number of class indicating in y. If NULL and tune=FALSE, it is the half of row number of x.
tune	A logical value indicating whether the best number of components should be tuned.
...	Arguments passed to or from other methods.
subset	An index vector specifying the cases to be used in the training sample.
na.action	A function to specify the action to be taken if NAs 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.

Details

A critical issue of applying linear discriminant analysis (LDA) is both the singularity and instability of the within-class scatter matrix. In practice, there are often a large number of features available, but the total number of training patterns is limited and commonly less than the dimension of the feature space. To tackle this issue, pcalda combines PCA and LDA for classification. It uses PCA for dimension reduction. The rotated data resulted from PCA will be the input variable to LDA for classification.

Value

An object of class pcalda containing the following components:

x	The rotated data on discriminant variables.
cl	The observed class labels of training data.
pred	The predicted class labels of training data.
posterior	The posterior probabilities for the predicted classes.
conf	The confusion matrix based on training data.
acc	The accuracy rate of training data.
ncomp	The number of principal components used for classification.

pca.out	The output of PCA.
lda.out	The output of LDA.
call	The (matched) function call.

Note

This function may be called giving either a formula and optional data frame, or a matrix and grouping factor as the first two arguments.

Author(s)

Wanchang Lin

See Also

predict.pcalda, plot.pcalda, tune.func

Examples

```
data(abr1)
cl   <- factor(abr1$fact$class)
dat  <- abr1$pos

## divide data as training and test data
idx <- sample(1:nrow(dat), round((2/3)*nrow(dat)), replace=FALSE)

## construct train and test data
train.dat  <- dat[idx,]
train.t    <- cl[idx]
test.dat   <- dat[-idx,]
test.t     <- cl[-idx]

## apply pcalda
model    <- pcalda(train.dat,train.t)
model
summary(model)

## plot
plot(model,dimen=c(1,2),main = "Training data",abbrev = TRUE)
plot(model,main = "Training data",abbrev = TRUE)

## confusion matrix
pred.te  <- predict(model, test.dat)$class
table(test.t,pred.te)
```

pcaplot *Plot Function for PCA with Grouped Values*

Description

Plot function for PCA with grouped values.

Usage

```
pcaplot(x, y, scale = TRUE, pcs = 1:2, ...)

pca.plot(x, y, scale=TRUE, abbrev = FALSE, ep.plot=FALSE,...)

pca.comp(x, scale=FALSE, pcs=1:2,...)
```

Arguments

x	A matrix or data frame to be plotted.
y	A factor or vector giving group information of columns of x.
scale	A logical value indicating whether the data set x should be scaled.
pcs	A vector of index of PCs to be plotted.
ep.plot	A logical value indicating whether the ellipse should be plotted.
abbrev	Whether the group labels are abbreviated on the plots. If abbrev > 0 this gives minlength in the call to abbreviate.
...	Further arguments to prcomp or lattice. See corresponding entry in xyplot for non-trivial details of lattice. For pcaplot, one argument is ep: an integer for plotting ellipse. 1 and 2 for plotting overall and group ellipse, respectively. Otherwise, none. For details, see panel.elli.1.

Value

pcaplot returns an object of class "trellis".

pca.comp returns a list with components:

scores	PCA scores
vars	Proportion of variance
varsn	A vector of string indicating the percentage of variance.

Note

Number of columns of x must be larger than 1. pcaplot uses lattice to plot PCA while pca.plot uses the basic graphics to do so. pca.plot plots PC1 and PC2 only.

Author(s)

Wanchang Lin

See Also

grpplot, panel.elli.1, pca_plot_wrap

Examples

```
## examples of 'pcaplot'
data(iris)
pcaplot(iris[,1:4], iris[,5],pcs=c(2,1),ep=2)
## change confidence interval (see 'panel.elli.1')
pcaplot(iris[,1:4], iris[,5],pcs=c(1,2),ep=2, conf.level = 0.9)

pcaplot(iris[,1:4], iris[,5],pcs=c(2,1),ep=1,
        auto.key=list(space="top", columns=3))
pcaplot(iris[,1:4], iris[,5],pcs=c(1,3,4))
tmp <- pcaplot(iris[,1:4], iris[,5],pcs=1:3,ep=2)
tmp

## change symbol's color, type and size
pcaplot(iris[,1:4], iris[,5],pcs=c(2,1),main="IRIS DATA", cex=1.2,
  auto.key=list(space="right", col=c("black","blue","red"), cex=1.2),
  par.settings = list(superpose.symbol = list(col=c("black","blue","red"),
                                              pch=c(1:3))))

## compare pcaplot and pca.plot.
pcaplot(iris[,1:4], iris[,5],pcs=c(1,2),ep=2)
pca.plot(iris[,1:4], iris[,5], ep.plot = TRUE)

## an example of 'pca.comp'
pca.comp(iris[,1:4], scale = TRUE, pcs=1:3)
```

---

plot.accest *Plot Method for Class 'accest'*

Description

Plot accuracy rate of each iteration.

Usage

```
## S3 method for class 'accest'
plot(x, main = NULL, xlab = NULL, ylab = NULL, ...)
```

Arguments

x	An object of class accest.
main	An overall title for the plot.
xlab	A title for the x axis.
ylab	A title for the y axis.
...	Additional arguments to the plot.

Details

This function is a method for the generic function `plot()` for class `accest`. It plots the accuracy rate against the index of iterations.

Value

Returns plot of class `accest`.

Author(s)

Wanchang Lin

See Also

`accest`

Examples

```
# Iris data
data(iris)
# Stratified cross-validation of PCALDA for Iris data
pars <- valipars(sampling="cv", niter=10, nreps=10, strat=TRUE)
acc  <- accest(Species~., data = iris, method = "pcalda", pars = pars)

acc
summary(acc)
plot(acc)
```

---

plot.maccest *Plot Method for Class 'maccest'*

Description

Plot accuracy rate with standard derivation of each classifier.

Usage

```
## S3 method for class 'maccest'
plot(x, main = NULL, xlab = NULL, ylab = NULL, ...)
```

Arguments

`x`	An object of class `maccest`.
`main`	An overall title for the plot.
`xlab`	A title for the x axis.
`ylab`	A title for the y axis.
`...`	Additional arguments to the plot.

Details

This function is a method for the generic function `plot()` for class `maccest`. It plots the accuracy rate with standard derivation against the classifiers.

Value

Returns plot of class `maccest`.

Author(s)

Wanchang Lin

See Also

`maccest`, `boxplot.maccest`

Examples

```
# Iris data
data(iris)
x      <- subset(iris, select = -Species)
y      <- iris$Species

method <- c("randomForest","svm","pcalda","knn")
pars   <- valipars(sampling="boot", niter = 10, nreps=4)
res    <- maccest(x, y, method=method, pars=pars,
                  comp="anova",kernel="linear")

res
plot(res)
```

`plot.pcalda` *Plot Method for Class 'pcalda'*

Description

Plot linear discriminants of `pcalda`.

Usage

```
## S3 method for class 'pcalda'
plot(x, dimen, ...)
```

Arguments

`x`	An object of class `pcalda`.
`dimen`	The index of linear discriminants to be used for the plot.
`...`	Further arguments. See corresponding entry in `xyplot` for non-trivial details. One argument is `ep`: an integer for plotting ellipse. `1` and `2` for plotting overall and group ellipse, respectively. Otherwise, none. For details, see `panel.elli.1`.

Details

This function is a method for the generic function `plot()` for class `pcalda`. If the length of `dimen` is greater than 2, a pairs plot is used. If the length of `dimen` is equal to 2, a scatter plot is drawn. Otherwise, the dot plot is drawn for the single component.

Value

An object of class `"trellis"`.

Author(s)

Wanchang Lin

See Also

`pcalda`, `predict.pcalda`, `lda_plot_wrap`,`panel.elli.1`.

Examples

```
data(abr1)
cl   <- factor(abr1$fact$class)
dat  <- abr1$pos

model <- pcalda(dat,cl)

## Second component versus first
plot(model,dimen=c(1,2),main = "Training data",ep=2)
## Pairwise scatterplots of several components
plot(model,main = "Training data",ep=1)

## The first component
plot(model,dimen=c(1),main = "Training data")
```

---

plot.plsc *Plot Method for Class 'plsc' or 'plslda'*

Description

Plot latent components of `plsc` or `plslda`.

Usage

```
## S3 method for class 'plsc'
plot(x, dimen, ...)

## S3 method for class 'plslda'
plot(x, dimen, ...)
```

Arguments

x	An object of class plsc or plslda.
dimen	The index of latent components to be used for the plot.
...	Further arguments. See corresponding entry in xyplot for non-trivial details. One argument is ep: an integer for plotting ellipse. 1 and 2 for plotting overall and group ellipse, respectively. Otherwise, none. For details, see panel.elli.1.

Details

Two functions are methods for the generic function plot() of class plsc and plslda.

If the length of dimen is greater than 2, a pairs plot is used. If the length of dimen is equal to 2, a scatter plot is drawn. Otherwise, the dot plot is drawn for the single component.

Value

An object of class "trellis".

Author(s)

Wanchang Lin

See Also

plsc, predict.plsc,plslda, predict.plslda, pls_plot_wrap, panel.elli.1.

Examples

```
data(abr1)
cl   <- factor(abr1$fact$class)
dat  <- abr1$pos

mod.plsc    <- plsc(dat,cl,ncomp=4)
mod.plslda  <- plslda(dat,cl,ncomp=4)

## Second component versus first
plot(mod.plsc,dimen=c(1,2),main = "Training data", ep = 2)
plot(mod.plslda,dimen=c(1,2),main = "Training data", ep = 2)

## Pairwise scatterplots of several components
plot(mod.plsc, main = "Training data", ep = 1)
plot(mod.plslda, main = "Training data", ep = 1)

## single component
plot(mod.plsc,dimen=c(1),main = "Training data")
plot(mod.plslda,dimen=c(1),main = "Training data")
```

plsc *Classification with PLSDA*

Description

Classification with partial least squares (PLS) or PLS plus linear discriminant analysis (LDA).

Usage

```
plsc(x, ...)

plslda(x, ...)

## Default S3 method:
plsc(x, y, pls="simpls",ncomp=10, tune=FALSE,...)

## S3 method for class 'formula'
plsc(formula, data = NULL, ..., subset, na.action = na.omit)

## Default S3 method:
plslda(x, y, pls="simpls",ncomp=10, tune=FALSE,...)

## S3 method for class 'formula'
plslda(formula, data = NULL, ..., subset, na.action = na.omit)
```

Arguments

formula	A formula of the form groups ~ x1 + x2 + ... 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 formula are preferentially to be taken.
x	A matrix or data frame containing the explanatory variables if no formula is given as the principal argument.
y	A factor specifying the class for each observation if no formula principal argument is given.
pls	A method for calculating PLS scores and loadings. The following methods are supported: • simpls: SIMPLS algorithm. • kernelpls: kernel algorithm. • oscorespls: orthogonal scores algorithm. For details, see simpls.fit, kernelpls.fit and oscorespls.fit in package **pls**.
ncomp	The number of components to be used in the classification.
tune	A logical value indicating whether the best number of components should be tuned.

...	Arguments passed to or from other methods.
subset	An index vector specifying the cases to be used in the training sample.
na.action	A function to specify the action to be taken if NAs 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.

Details

plcs implements PLS for classification. In details, the categorical response vector y is converted into a numeric matrix for regression by PLS and the output of PLS is convert to posteriors by softmax method. The classification results are obtained based on the posteriors. plslda combines PLS and LDA for classification, in which, PLS is for dimension reduction and LDA is for classification based on the data transformed by PLS.

Three PLS functions,simpls.fit, kernelpls.fit and oscorespls.fit, are implemented in package **pls**.

Value

An object of class plsc or plslda containing the following components:

x	A matrix of the latent components or scores from PLS.
cl	The observed class labels of training data.
pred	The predicted class labels of training data.
conf	The confusion matrix based on training data.
acc	The accuracy rate of training data.
posterior	The posterior probabilities for the predicted classes.
ncomp	The number of latent component used for classification.
pls.method	The PLS algorithm used.
pls.out	The output of PLS.
lda.out	The output of LDA used only by plslda.
call	The (matched) function call.

Note

Two functions may be called giving either a formula and optional data frame, or a matrix and grouping factor as the first two arguments.

Author(s)

Wanchang Lin

References

Martens, H. and Nas, T. (1989) *Multivariate calibration.* John Wiley & Sons.

See Also

kernelpls.fit, simpls.fit, oscorespls.fit, predict.plsc, plot.plsc, tune.func

Examples

```
library(pls)
data(abr1)
cl   <- factor(abr1$fact$class)
dat  <- preproc(abr1$pos , y=cl, method=c("log10"),add=1)[,110:500]

## divide data as training and test data
idx <- sample(1:nrow(dat), round((2/3)*nrow(dat)), replace=FALSE)

## construct train and test data
train.dat  <- dat[idx,]
train.t    <- cl[idx]
test.dat   <- dat[-idx,]
test.t     <- cl[-idx]

## apply plsc and plslda
(res   <- plsc(train.dat,train.t, ncomp = 20, tune = FALSE))
## Estimate the mean squared error of prediction (MSEP), root mean squared error
## of prediction (RMSEP) and R^2 (coefficient of multiple determination) for
## fitted PLSR model
MSEP(res$pls.out)
RMSEP(res$pls.out)
R2(res$pls.out)

(res.1  <- plslda(train.dat,train.t, ncomp = 20, tune = FALSE))
## Estimate the mean squared error of prediction (MSEP), root mean squared error
## of prediction (RMSEP) and R^2 (coefficient of multiple determination) for
## fitted PLSR model
MSEP(res.1$pls.out)
RMSEP(res.1$pls.out)
R2(res.1$pls.out)

## Not run:
## with function of tuning component numbers
(z.plsc   <- plsc(train.dat,train.t, ncomp = 20, tune = TRUE))
(z.plslda <- plslda(train.dat,train.t, ncomp = 20, tune = TRUE))

## check nomp tuning results
z.plsc$ncomp
plot(z.plsc$acc.tune)
z.plslda$ncomp
plot(z.plslda$acc.tune)

## plot
plot(z.plsc,dimen=c(1,2,3),main = "Training data",abbrev = TRUE)
plot(z.plslda,dimen=c(1,2,3),main = "Training data",abbrev = TRUE)

## predict test data
```

```
pred.plsc   <- predict(z.plsc, test.dat)$class
pred.plslda <- predict(z.plslda, test.dat)$class

## classification rate and confusion matrix
cl.rate(test.t, pred.plsc)
cl.rate(test.t, pred.plslda)


## End(Not run)
```

`predict.osc` *Predict Method for Class 'osc'*

Description

Pre-processing of new data by `osc`.

Usage

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

Arguments

`object`	Object of class `osc`.
`newdata`	A matrix or data frame of cases to be corrected by OSC.
`...`	Arguments based from or to other methods.

Details

This function is a method for the generic function `predict()` for class `osc`. If `newdata` is omitted, the corrected data set used in model of `osc` will be returned.

Value

A list containing the following components:

`x`	A matrix of OSC corrected data set.
`Q2`	The fraction of variation in X after OSC correction for the new data.

Author(s)

Wanchang Lin

See Also

`osc`, `osc_wold`, `osc_sjoblom`, `osc_wise`

Examples

```
data(abr1)
cl   <- factor(abr1$fact$class)
dat  <- abr1$pos

## divide data as training and test data
idx <- sample(1:nrow(dat), round((2/3)*nrow(dat)), replace=FALSE)

## construct train and test data
train.dat  <- dat[idx,]
train.t    <- cl[idx]
test.dat   <- dat[-idx,]
test.t     <- cl[-idx]

## build OSC model based on the training data
res <- osc(train.dat, train.t, method="wold",osc.ncomp=2, pls.ncomp=4)
names(res)
res
summary(res)

## pre-process test data by OSC
test <- predict(res,test.dat)
test.dat.1 <- test$x
```

`predict.pcalda` *Predict Method for Class 'pcalda'*

Description

Prediction of test data using `pcalda`.

Usage

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

Arguments

`object`	Object of class `pcalda`.
`newdata`	A matrix or data frame of cases to be classified.
`...`	Arguments based from or to other methods.

Details

This function is a method for the generic function `predict()` for class `pcalda`. If `newdata` is omitted, the results of training data in `pcalda` object will be returned.

Value

A list with components:

class	The predicted class (a factor).
posterior	The posterior probabilities for the predicted classes.
x	The rotated test data by the projection matrix of LDA.

Author(s)

Wanchang Lin

See Also

pcalda, plot.pcalda

Examples

```
data(iris3)

tr    <- sample(1:50, 25)
train <- rbind(iris3[tr,,1], iris3[tr,,2], iris3[tr,,3])
test  <- rbind(iris3[-tr,,1], iris3[-tr,,2], iris3[-tr,,3])
cl    <- factor(c(rep("s",25), rep("c",25), rep("v",25)))

z     <- pcalda(train, cl)
pred  <- predict(z, test)

## plot the projected data.
grpplot(pred$x, pred$class, main="PCALDA: Iris")
```

---

predict.plsc *Predict Method for Class 'plsc' or 'plslda'*

Description

Prediction of test data using plsc or plslda.

Usage

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

Arguments

object	Object of class plsc or plslda.
newdata	A matrix or data frame of cases to be classified.
...	Arguments based from or to other methods.

Details

Two functions are methods for the generic function `predict()` for class `plsc` or `plslda`. If `newdata` is omitted, the results of training data in `plsc` or `plslda` object will be returned.

Value

A list with components:

`class`	The predicted class (a factor).
`posterior`	The posterior probabilities for the predicted classes.
`x`	The rotated test data by the projection matrix of PLS.

Author(s)

Wanchang Lin

See Also

`plsc`, `plot.plsc`,`plslda`, `plot.plslda`

Examples

```
data(iris3)

tr    <- sample(1:50, 25)
train <- rbind(iris3[tr,,1], iris3[tr,,2], iris3[tr,,3])
test  <- rbind(iris3[-tr,,1], iris3[-tr,,2], iris3[-tr,,3])
cl    <- factor(c(rep("s",25), rep("c",25), rep("v",25)))

## model fit using plsc and plslda without tuning of ncomp
(z.plsc       <- plsc(train, cl))
(z.plslda     <- plslda(train, cl))
## predict for test data
pred.plsc    <- predict(z.plsc, test)
pred.plslda  <- predict(z.plslda, test)

## plot the projected test data.
grpplot(pred.plsc$x, pred.plsc$class, main="PLSC: Iris")
grpplot(pred.plslda$x, pred.plslda$class, main="PLSLDA: Iris")
```

`preproc` *Pre-process Data Set*

Description

Pre-process a data frame or matrix by different methods.

Usage

```
preproc (x, y=NULL,method="log",add=1)

preproc.sd(x, y=NULL, na.rm = FALSE)

preproc.const(x, y, tol = 1.0e-4)
```

Arguments

x	A numeric data frame or matrix to be pre-processed.
y	A factor specifying the group. It is only used by the method TICnorm in preproc.
method	A method used to pre-process the data set. The following methods are supported: • center: Centering • auto: Auto scaling • range: Range scaling • pareto: Pareto scaling • vast: Vast scaling • level: Level scaling • log: Log transformation (default) • log10: Log 10 transformation • sqrt: Square root transformation • asinh: Inverse hyperbolic sine transformation • TICnorm: TIC normalisation
na.rm	A logical value indicating whether NA values should be stripped before the computation proceeds.
add	A numeric value for addition used in the logarmath transformation log and log10.
tol	A tolerance to decide if a matrix is singular; it will reject variables and linear combinations of unit-variance variables whose variance is less than tol^2.

Details

preproc transforms data set by provided method.

preproc.sd removes variables which have (near) zero S.D with or without respect to class/grouped information.

preproc.const removes variables appears to be constant within groups / classes.

Value

A pre-processed data set.

Author(s)

Wanchang Lin

References

Berg, R., Hoefsloot, H., Westerhuis, J., Smilde, A. and Werf, M. (2006), Centering, scaling, and transformations: improving the biological information content of metabolomics data, *BMC Genomics*, 7:142

Examples

```
data(abr1)
cl   <- factor(abr1$fact$class)
dat  <- abr1$pos

## normalise data set using "TICnorm"
z.1  <- preproc(dat, y=cl, method="TICnorm")

## scale data set using "log10"
z.2 <- preproc(dat,method="log10", add=1)

## or scale data set using "log" and "TICnorm" sequentially
z.3 <- preproc(dat,method=c("log","TICnorm"), add=0.1)
```

`pval.util` *P-values Utilities*

Description

Functions to handle p-values of data set.

Usage

```
pval.test(x,y, method="oneway.test",...)

pval.reject(adjp,alpha)
```

Arguments

`x`	A data frame or matrix of data set.
`y`	A factor or vector of class.
`method`	Hypothesis test such as `t.test` and `wilcox.test`.
`adjp`	A matrix-like p-values of simultaneously testing.
`alpha`	A vector of cutoff of p-values or Type I error rate.
`...`	Arguments to pass.

Value

`pval.test` returns a vector of p-values.

`pval.reject` returns a matrix indicating rejected number according to cutoff.

Author(s)

Wanchang Lin

Examples

```
library(lattice)

## Example for pval.test and pval.reject
## prepare data set
data(abr1)
cls <- factor(abr1$fact$class)
dat <- abr1$pos[,200:500]
dat <- preproc(dat, method="log")

## select class "1" and "2" for feature ranking
ind <- grepl("1|2", cls)
dat <- dat[ind,,drop=FALSE]
cls <- cls[ind, drop=TRUE]

## univariate p-values and its adjusted p-values
pval <- sort(pval.test(dat, cls, method="t.test"))

## adjust p-values
pval.ad <- sapply(c("fdr","bonferroni","BY"), function(y){
  p.adjust(pval, method=y)
})
pval.ad <- cbind(raw=pval, pval.ad)
pval.reject(pval.ad,c(0.005, 0.01, 0.05))

## plot the all p-values
tmp <- cbind(pval.ad, idx=1:nrow(pval.ad))
tmp <- data.frame(tmp)

# pval_long <- melt(tmp, id="idx")
pval_long <- data.frame(idx = tmp$idx, stack(tmp, select = -idx))
pval_long <- pval_long[c("idx", "ind", "values")]
names(pval_long) <- c("idx", "variable", "value")

pval.p <- xyplot(value~idx, data=pval_long, groups=variable,
                 par.settings = list(superpose.line = list(lty=c(1:7))),
                 as.table = TRUE, type="l",
                 par.strip.text = list(cex=0.65), ylim=c(-0.005, 1.0),
                 ylab="P-values", xlab="Index of variables",
                 main="p-values",
                 auto.key = list(lines=TRUE, points = FALSE,space="right"),
                 panel = function(x, y,...) {
                   panel.xyplot(x, y, ...)
                   panel.abline(h = 0.05, col = "red",lty =2)
                 })
pval.p
```

save.tab *Save List of Data Frame or Matrix into CSV File*

Description

Save a list of data frame or matrix into a CSV file

Usage

```
save.tab(x, filename="temp.csv", firstline="\n")
```

Arguments

x	A list of data frame or matrix.
filename	A character string for saved file name.
firstline	A string giving some description of the saved file.

Details

This function gives a quick option to save a set of data frame or matrix into a single table file.

Value

No return value, called for side effects.

Author(s)

Wanchang Lin

See Also

write.table

Examples

```
## Not run:
data(abr1)
dat <- preproc(abr1$pos[,200:400], method="log10")
cls <- factor(abr1$fact$class)

tmp <- dat.sel(dat, cls, choices=c("1","2"))
x   <- tmp[[1]]$dat
y   <- tmp[[1]]$cls

fs.method <- c("fs.anova","fs.rf","fs.rfe")
fs.pars   <- valipars(sampling="cv",niter=10,nreps=5)
fs <- feat.mfs(x, y, fs.method, fs.pars)   ## with resampling
names(fs)
```

```
fs <- fs[1:3]

## save consistency of feature selection
filename  <- "fs.csv"
firstline <- paste('\nResults of feature selection', sep='')

save.tab(fs, filename, firstline)

## End(Not run)
```

`stats.util` *Statistical Summary Utilities for Two-Classes Data*

Description

Functions to summarise two-group data.

Usage

```
stats.vec(x,y, method="mean",test.method = "wilcox.test",fc=TRUE,...)
stats.mat(x,y, method="mean",test.method = "wilcox.test",
          padj.method= "fdr",fc=TRUE,...)
```

Arguments

`x`	A data frame or matrix of data set for `stats.mat` or data vector for `stats.vec`.
`y`	A factor or vector of class. Two classes only.
`method`	method for group center such as `mean` or `median`.
`test.method`	method for p-values from parametric test such as `t.test` or non-parametric test such as `wilcox.test`.
`padj.method`	method for p-values correction. Can be one in `p.adjust.methods`: "holm", "hochberg", "hommel", "bonferroni", "BH", "BY", "fdr" and "none".
`fc`	a flag for fold-change.
`...`	Additional parameters.

Value

`stats.vec` returns an vector of center, group center, fold change, log2 fold change, AUC and p-value.

`stats.mat`, which is an wrapper function of `stats.vec`, returns an data frame of center, group center, fold change, log2 fold-change, AUC, p-value and adjusted p-values.

Note

If `x` has negative values, the results of fold-change and log2 fold-change are not reliable.

Author(s)

Wanchang Lin

Examples

```
data(iris)
sel <- c("setosa", "versicolor")
grp <- iris[,5]
idx <- match(iris[,5],sel,nomatch = 0) > 0

cls <- factor(iris[idx,5])
dat <- iris[idx,1:4]

## stats for the single vector
stats.vec(dat[[1]],cls, method = "median",test.method = "wilcox.test")

## use matrix format
tab <- stats.mat(dat,cls, method = "median",test.method = "wilcox.test",
                 padj.method = "BH")
sapply(tab, class)
```

---

trainind *Generate Index of Training Samples*

Description

Generate index of training samples. The sampling scheme includes leave-one-out cross-validation (`loocv`), cross-validation (`cv`), randomised validation (`random`) and bootstrap (`boot`).

Usage

```
trainind(cl, pars = valipars())
```

Arguments

`cl`	A factor or vector of class.
`pars`	A list of sampling parameters for generating training index. It has the same structure as the output of `valipars`. See `valipars` for details.

Value

Returns a list of training index.

Author(s)

Wanchang Lin

See Also

valipars

Examples

```
## A trivia example
x <- as.factor(sample(c("a","b"), 20, replace=TRUE))
table(x)
pars <- valipars(sampling="rand", niter=2, nreps=4, strat=TRUE,div=2/3)
(temp <- trainind(x,pars=pars))
(tmp  <- temp[[1]])
x[tmp[[1]]];table(x[tmp[[1]]])     ## train idx
x[tmp[[2]]];table(x[tmp[[2]]])
x[tmp[[3]]];table(x[tmp[[3]]])
x[tmp[[4]]];table(x[tmp[[4]]])

x[-tmp[[1]]];table(x[-tmp[[1]]])   ## test idx
x[-tmp[[2]]];table(x[-tmp[[2]]])
x[-tmp[[3]]];table(x[-tmp[[3]]])
x[-tmp[[4]]];table(x[-tmp[[4]]])

# iris data set
data(iris)
dat <- subset(iris, select = -Species)
cl  <- iris$Species

## generate 5-fold cross-validation samples
cv.idx <- trainind(cl, pars = valipars(sampling="cv", niter=2, nreps=5))

## generate leave-one-out cross-validation samples
loocv.idx <- trainind(cl, pars = valipars(sampling = "loocv"))

## generate bootstrap samples with 25 replications
boot.idx <- trainind(cl, pars = valipars(sampling = "boot", niter=2,
                                         nreps=25))

## generate randomised samples with 1/4 division and 10 replications.
rand.idx <- trainind(cl, pars = valipars(sampling = "rand", niter=2,
                                         nreps=10, div = 1/4))
```

---

tune.func *Functions for Tuning Appropriate Number of Components*

Description

Tune appropriate number of components (ncomp) for plsc, plslda or pcalda.

Usage

```
tune.plsc(x,y, pls="simpls",ncomp=10, tune.pars,...)

tune.plslda(x,y, pls="simpls",ncomp=10, tune.pars,...)

tune.pcalda(x,y, ncomp=NULL, tune.pars,...)
```

Arguments

x	A matrix or data frame containing the explanatory variables if no formula is given as the principal argument.
y	A factor specifying the class for each observation if no formula principal argument is given.
pls	A method for calculating PLS scores and loadings. The following methods are supported: • simpls: SIMPLS algorithm. • kernelpls: kernel algorithm. • oscorespls: orthogonal scores algorithm. For details, see simpls.fit, kernelpls.fit and oscorespls.fit in package **pls**.
ncomp	The number of components to be used in the classification.
tune.pars	A list of parameters using by the resampling method. See valipars for details.
...	Further parameters passed to tune.

Value

A list including:

ncomp	The best number of components.
acc.tune	Accuracy rate of components.

Author(s)

Wanchang Lin

See Also

plsc, plslda, pcalda,valipars

Examples

```
## Not run:
data(abr1)
cl   <- factor(abr1$fact$class)
dat  <- preproc(abr1$pos , y=cl, method=c("log10"),add=1)[,110:500]

## divide data as training and test data
```

```
idx <- sample(1:nrow(dat), round((2/3)*nrow(dat)), replace=FALSE)

## construct train and test data
train.dat  <- dat[idx,]
train.t    <- cl[idx]
test.dat   <- dat[-idx,]
test.t     <- cl[-idx]

## tune the best number of components
ncomp.plsc   <- tune.plsc(dat,cl, pls="simpls",ncomp=20)
ncomp.plslda <- tune.plslda(dat,cl, pls="simpls",ncomp=20)
ncomp.pcalda <- tune.pcalda(dat,cl, ncomp=60)

## model fit
(z.plsc   <- plsc(train.dat,train.t, ncomp=ncomp.plsc$ncomp))
(z.plslda <- plslda(train.dat,train.t, ncomp=ncomp.plslda$ncomp))
(z.pcalda <- pcalda(train.dat,train.t, ncomp=ncomp.pcalda$ncomp))

## or indirect use tune function in model fit
z.plsc   <- plsc(train.dat,train.t, ncomp=20, tune=TRUE)
z.plslda <- plslda(train.dat,train.t, ncomp=20, tune=TRUE)
z.pcalda <- pcalda(train.dat,train.t, ncomp=60, tune=TRUE)

## predict test data
pred.plsc   <- predict(z.plsc, test.dat)$class
pred.plslda <- predict(z.plslda, test.dat)$class
pred.pcalda <- predict(z.pcalda, test.dat)$class

## classification rate and confusion matrix
cl.rate(test.t, pred.plsc)
cl.rate(test.t, pred.plslda)
cl.rate(test.t, pred.pcalda)


## End(Not run)
```

`valipars` *Generate Control Parameters for Resampling*

Description

Generate the control parameters for resampling process.

Usage

```
valipars(sampling="cv", niter=10, nreps=10, strat=FALSE,div = 2/3)
```

Arguments

sampling	Sampling scheme. Valid options are: • loocv. Leave-one-out cross-validation • cv. Cross-validation (default) • rand. Randomised validation (holdout) • boot. Bootstrap
niter	Number of iteration or repeat for validation.
nreps	Number of replications in each iteration.
strat	A logical value indicating whether the stratification should be applied to cv, rand and boot.
div	Proportion of data used for training in randomised validation method.

Details

valipars provides a list of control parameters for the resampling or validation in the process of accuracy evaluation or feature selection process.

Value

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

Author(s)

Wanchang Lin

See Also

trainind

Examples

```
## generate control parameters for the re-sampling scheme with 5-fold
## cross-validation and iteration of 10 times
valipars(sampling = "cv", niter = 10, nreps = 5)

## generate control parameters for the re-sampling scheme with
## 25-replication bootstrap and iteration of 100 times
valipars(sampling = "boot", niter = 100, nreps = 25,strat=TRUE)

## generate control parameters for the re-sampling scheme with
## leave-one-out cross-validation
valipars(sampling = "loocv")
```

Index