# Package 'FIEmspro'

August 18, 2010

**Version** 1.1-0

**Date** 2007-10-16

**Title** Flow Injection Electrospray Mass Spectrometry Processing: \{ }\{ } data processing, classification modelling and variable selection in metabolite fingerprinting

**Depends** e1071, randomForest, MASS, ncdf, impute, KernSmooth,RColorBrewer, xtable, plotrix

**Author** Manfred Beckmann, David Enot and Wanchang Lin

**Maintainer** Wanchang Lin <wll@aber.ac.uk>

**Description** Functions of data processing for metabolomics.

**License** GPL version 2.

**SaveImage** no

**LazyLoad** yes

## R topics documented:

---

`abr1` *abr1 dataset*

---

## Description

Real world FIE-MS dataset.

## Usage

```
data(abr1)
```

## Details

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 processed using `fiems_ltq_main`. Both positive and negative ion mode are given (`abr1$pos` and `abr1$neg`). To avoid confusions, variable names are given with a letter corresponding to the ionisation mode followed by the actual nominal mass value (e.g. P130 corresponds to the nominal mass 130 in the positive mode).

Experimental factors are given in the `abr1$fact` data frame:

- `injorder`: sample injection order
- `name`: sample name
- `rep`: biological replicate for a given class
- `day`: number of days following infection after which the sample has been harvested - Level H corresponds to an healthy plant.
- `class`: identical to day except that `class=6` when `day=H`
- `pathcdf,filecdf,name.org,remark`: are generated from profile processing and are kept for traceability purposes.

Factor of interest for classification are contained in `abr1$fact$day`. There are 20 biological replicates in each class has

## 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. |

## Author(s)

Manfred Beckmann, David Enot and Wanchang Lin <meb,dle, wll@aber.ac.uk>

## Examples

```
# Load data set
data(abr1)

# Select data set
dat <- abr1$neg

# number of observations and variables in the negative mode matrix
dim(dat)

# names of the variables
dimnames(dat)[[2]]

# print out the experimental factors
print(abr1$fact)

# check out the repartition of class
table(abr1$fact$class)
```

---

# accest *Classification Wrapper Using Customised Classifiers*

---

## Description

Wrapper function for calculating classification estimates using pre-defined data partitioning sets (valipars and trainind). This function works with two type of classifiers. First generic classifiers that fulfil R standards to define predictive techniques such as the ones available in packages like **MASS**, **e1071** or randomForest and nlda are normally handle with accest: the name of function (clmeth in the accest call) must be accompanied with an S3 method predict; the later function should return a list with component 'class' (hard classification) and if possible 'prob' or 'posterior' for class probabilities. If the algorithm doesn't fulfil these requirements, two postions can be adopted: 1) define explicitly the algorithm so that it means R standards 2) define customised a function that returns necessary informations. The second ('quicky and dirty') approach is illustrated in an example given below. Unless the classifier can only cope with two-class tasks, this function allows the manipulation of any problem complexity. Three types of estimates are given for each replication: accuracy, so-called margin and AUC (see details). Data input can be in the form of data matrix + class vector, following the classic formula type or derived from dat.sel1.

## Usage

```
accest(...)

## Default S3 method:
accest(dat, cl, clmeth, pars = NULL, tr.idx = NULL, verb=TRUE, clmpi=NULL, seed=

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

## S3 method for class 'dlist':
accest(dlist,  clmeth,pars = NULL, tr.idx = NULL, ...)
```

## 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. |
| dlist | A matrix or data frame containing the explanatory variables if no formula is given as the principal argument. |
| 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. |
| clmeth | Classifier function. For details, see `note` below. |
| pars | A list of parameters using by the resampling method such as *Leave-one-out cross-validation*, *Cross-validation*, *Bootstrap* and *Randomised validation (hold-out)*. See `valipars` for details. |
| tr.idx | User defined index of training samples. Can be generated by `trainind`. |
| verb | Should iterations be printed out? |
| clmpi | snow cluster information |
| seed | Seed. |
| ... | Additional parameters to be passed to `clmeth`. |
| 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

Seexxxx for common details.

## Value

An object of class `accest`, including the components:

| | |
|---|---|
| clmeth | Classification method used. |
| acc | Average accuracy. |

| | |
|---|---|
| acc.iter | Accuracy at each iteration. |
| acc.std | Standard derivation of accuracy. |
| mar | Average predictive margin. |
| mar.iter | Predictive margin of each iteration. |
| auc | Average area under receiver operating curve (AUC). |
| auc.iter | AUC of each iteration. |
| sampling | Sampling scheme used. |
| niter | Number of iterations. |
| nreps | Number of replications at each iteration. |
| acc.boot | Detailed bootstrap accuracy estimates when bootstrap validation method is employed. |
| argfct | Arguments passed to the classifier. |
| pred.all | For each iteration, list of the fold/bootstrap id and the true and predicted classes. |
| cl.task | Discrimination task. |
| mod | List of information return by the user defined classifier function. |

## Author(s)

David Enot <dle@aber.ac.uk> and Wanchang Lin <wll@aber.ac.uk>

## See Also

valipars, trainind

## Examples

```
## ------------------------------------------------------------------
## simple customised function
## sameasrf simply reproduces the RF modelling task
sameasrf <- function(data,...){
  dots <- list(...)

  ## Build RF model and predict dat.te
  mod  <- randomForest(data$tr,data$cl,...)
  ## Soft predictions (optional if ROC/margin analyses required)
  prob <- predict(mod,data$te,type="vote")
  ## Hard predictions
  pred <- predict(mod,data$te)

  # For illustration, mod does not contain anything
  list(mod=NULL,pred=pred,prob=prob,arg=dots)
}

## ------------------------------------------------------------------
## compare accest with randomForest
## and sameasrf
data(iris)
dat=as.matrix(iris[,1:4])
cl=as.factor(iris[,5])
pars   <- valipars(sampling = "boot",niter = 2, nreps=10)
```

```
tr.idx <- trainind(iris$Species,pars)

set.seed(71)
acc.1 <- accest(dat,cl, clmeth = "sameasrf",
                  pars = pars,tr.idx = tr.idx,ntree = 200)
summary(acc.1)

set.seed(71)
acc.2 <- accest(dat,cl, clmeth = "randomForest",
                  pars = pars,tr.idx = tr.idx,ntree = 200)
summary(acc.2)

### compare acc.1 and acc.2 bootstrap error estimates
print(acc.1$acc.boot-acc.2$acc.boot)

########################################
## Try formula type
set.seed(71)
acc.3 <- accest(Species~., data = iris, clmeth = "randomForest",
                  pars = pars,tr.idx = tr.idx,ntree = 200)
summary(acc.3)

## Try dlist type from dat.sel1
set.seed(71)
dat2=dat.sel1(dat,cl,pars=pars)
acc.4 <- accest(dat2[[1]], clmeth = "randomForest",
                  pars = pars,tr.idx = tr.idx,ntree = 200)
summary(acc.4)
```

---

`dat.sel` *Generate Pairwise Data Set Based on Class Labels*

---

## Description

Generate pairwise data set based on class labels.

## Usage

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

## Arguments

| | |
|---|---|
| `dat` | A data frame or matrix. |
| `cl` | A factor or vector of class. |
| `choices` | The vector or list of class labels to be chosen for binary classification. |

## Details

This function is used to provide the data set for the binary combination of the class factor. If `choices` is `NULL`, the binary combination of for all class labels will be done. If `choices` has one class label, the comparisons between this one and any other class are done. If `choices` has more than three class lables, enumerate the combinations or permutations of the elements of `choices`. For details, see `examples` below.

## Value

A list with components:

| | |
|---|---|
| dat | Pairwise data set. |
| cl | Pairwise class label. |
| com | A matrix of the combinations or permutations of the elements of pairwise vector. |

## Author(s)

Wanchang Lin <wll@aber.ac.uk>

## 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")))

## 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")))

## 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))
```

---

# dat.sel1 *Generate Data Set List*

---

## Description

Generate subset data sets based on class labels. This function allows generation of a selection of pairwise problems and/or multiple class problems. The main objective of this function is to alleviate generation of several variables containing data matrices, class information or validation strategy. Each subset can enter accest and feat.rank.re analysis without specifying both class and data matrix. Additionally, each subset can also contain validation parameters so that direct comparison between classifiers and feature ranking technique can be easily done.

## Usage

```
dat.sel1(dat, cl, pwise = NULL, mclass = list(),pars=NULL)
```

## Arguments

| | |
|---|---|
| dat | A data frame or matrix. |
| cl | A factor or vector of class. |
| pwise | The vector or list of class labels to be chosen for binary classification. |

| | |
|---|---|
| mclass | The vector or list of class labels to be chosen to be included in a multiple classification task. |
| pars | Partitioning information. |

## Details

This function is used to provide the data set for the binary combination of the class factor. If pwise is list(), the binary combination of for all class labels will be done. If pwise has one class label, the comparisons between this one and any other class are done. If pwise has more than three class lables, enumerate the combinations or permutations of the elements of pwise. For details, see examples below.

## Value

A list with components:

| | |
|---|---|
| nam | Discrimination task. |
| dat | Subset of the dataset. |
| cl | Class labels. |
| pars | Object of type pars. |
| tr.idx | Object of type trainind. |
| lsamp | Sample ids in the original data matrix. |

## Author(s)

David <dle@aber.ac.uk>

## Examples

```
data(abr1)
x<-abr1$pos[,110:120]
y<-factor(abr1$fact$day)

## generate data set with all pairwise containing class "1"
dat1 <- dat.sel1(x,y,pwise="1",mclass=NULL)
unlist(lapply(dat1,function(x) x$name))

## generate data set with pairwise between classes "1" and "H"
dat1 <- dat.sel1(x,y,pwise=c("1","H"),mclass=NULL)
unlist(lapply(dat1,function(x) x$name))

## generate data set with pairwise between classes "1", "2", "H"
dat1 <- dat.sel1(x,y,pwise=c("2","1","H"),mclass=NULL)
unlist(lapply(dat1,function(x) x$name))

## generate data set with all pairwises containing "1" and "H"
dat1 <- dat.sel1(x,y,pwise=list("1","H"),mclass=NULL)
unlist(lapply(dat1,function(x) x$name))

## generate data set with 3 classes "1", "2" and "H"
dat1 <- dat.sel1(x,y,pwise=NULL,mclass=c("1","2","H"))
unlist(lapply(dat1,function(x) x$name))

## generate data set with all pairwises containing "1" and "H"
```

```
## on which partitioning is a 1*5CV
pars=valipars(sampling = "cv", niter = 1, nreps = 5)
dat1 <- dat.sel1(x,y,pwise=list("1","H"),mclass=NULL, pars=pars)
unlist(lapply(dat1,function(x) x$name))

print(dat1[[1]])
```

---

`feat.rank.re` *Wrapper for Resampling Based Feature Ranking*

---

## Description

Wrapper for performing feature ranking method on multiple subsets of the original data. At each iteration, only the training samples defined in `tr.idx` (or optionally `pars` only) are used to rank the variables. Features rank, order and saliency indicators calculated on the whole data are also given in the output. As for `accest` this function allows the use of multiple processors as long as the cluster has been set up with the **snow** package. Data input can be in the form of `data` matrix + `class` vector, following the classic formula type or derived from `dat.sel1`.

## Usage

```
  feat.rank.re(...)

  ## Default S3 method:
feat.rank.re(x,y,method,pars = valipars(),tr.idx=NULL,clmpi=NULL, seed=NULL, ...

  ## S3 method for class 'formula':
feat.rank.re(formula, data = NULL, ...)

 ## S3 method for class 'dlist':
feat.rank.re(dlist,  method, pars = NULL, tr.idx = NULL, ...)
```

## 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. |
| `x` | A matrix or data frame containing the explanatory variables. |
| `dlist` | A matrix or data frame containing the explanatory variables if no formula is given as the principal argument. |
| `data` | Data frame from which variables specified in `formula` are preferentially to be taken. |
| `y` | A factor specifying the class for each observation. |
| `method` | Feature ranking method to be used. See `fs.techniques` for details. |
| `pars` | A list of resampling scheme or validation 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 of type `trainind`. Generated by `trainind` if `tr.idx=NULL`. |

| | |
|---|---|
| clmpi | snow cluster information |
| seed | Seed. |
| ... | Additional parameters to be passed to method. See fs.techniques for details. |

## Details

The structure of the feat.rank.re object is as follows:

**method** Feature ranking method used.

**fs.rank** A vector of final feature ranking scores.

**fs.order** A vector of final feature order from best to worst.

**fs.stats** A vector of means of statistics or measurements in all computation.

**rank.list** Feature rank list of all computation.

**order.list** Feature order list of all computation.

**pars** Resampling parameters.

**tr.idx** Index of training samples.

**pars.min** Condensed form of the resampling strategy for output purposes.

**cl.task** Condensed form of the classification task.

**all** Feature ranking object originated from the overall dataset.

## Value

feat.rank.re object.

## Author(s)

David Enot <dle@aber.ac.uk>

## See Also

valipars, ftrank.agg, fs.techniques

## Examples

```
## load abr1
data(abr1)
y   <- factor(abr1$fact$class)
x <- preproc(abr1$pos , y=y, method=c("log10","TICnorm"),add=1)[,110:500]
## Select classes 1 and 2
pars   <- valipars(sampling="boot",niter=2,nreps=5)
dat <- dat.sel1(x, y, pwise=c("1","2"),mclass=NULL,pars=pars)

## multiple rankings using AUC
z      <- feat.rank.re(dat[[1]],method="fs.auc")

## print content of z
names(z)
```

fiems_lct_main *LCT Mass Binning*

## Description

Main Routine for 'Mass Binning' to nominal mass and 'Mass Spectrum' generation in high-throughput Flow Injection Electrospray Ionisation Mass Spectrometry (FIE-MS). This routine reads ANDI NetCDF files (`*.cdf`) of LCT/Q-ToF `*.raw` data files converted in the DataBridge program (Dbridge, MassLynx, Micromass).

## Usage

```
fiems_lct_main(my_path,runinfo,y1,y2,y3,y4,limit=0.82,
               save.file=TRUE,file.name="LCT-mean.RData")
```

## Arguments

| | |
|---|---|
| `my_path` | A character string indicating the working directory where `runinfo.csv` file and folder containing `*.cdf`-files are located. |
| `runinfo` | A `*.csv` file containing at least the following run information (header row): `pathcdf` and `filecdf` . For details, see the description in `Examples` below. |
| `y1` | A numeric value used for mass spectrum generation: start scan 'sample'. For details, see the description in `Examples` below. |
| `y2` | A numeric value used for mass spectrum generation: end scan 'sample'. For details, see the description in `Examples` below. |
| `y3` | A numeric value used for mass spectrum generation: start scan 'background'. For details, see the description in `Examples` below. |
| `y4` | A numeric value used for mass spectrum generation: end scan 'background'. For details, see the description in `Examples` below. |
| `limit` | A numeric value defining the rounding limit for binning m/z-values to nominal mass. |
| `save.file` | A logical value indicating whether or not to save the results (default is `TRUE`). |
| `file.name` | A character for saved file name if `save.file` is `TRUE`. |

## Details

This routine is designed to handle only one MassLynx (Micromass) specific function (e.g. data acquired at one cone voltage). Each `*.cdf`-file will result in one mass spectrum. Principle in brief: load `*.cdf`-file (`pathcdf` and `filecdf` information in `runinfo.csv`); bin m/z-values to nominal mass between 'limit-1' and 'limit'; sum up intensities of binned m/z values; generate sample matrix 'smat' between scans y1 and y2 and background matrix 'bmat' between scans y3 and y4; subtract: mat=smat-bmat; calculate mean of resulting matrix 'mat'; potential negative values are set to 'zero'. The implemented timer-function should be accurate for up to 24 hours which could comprise more than 7000 `*.cdf`-files per experiment.

## Value

A list containing the following components:

| | |
|---|---|
| `mat` | Single matrix [runs x nominal masses] of the full mass range [0:2000] in the ionisation mode. |
| `runinfo` | Same as argument stored for reference purposes. Additional information for each run like sample name or class can be used for further analysis (e.g. nlda). |
| `scrng` | A vector of `y1`, `y2`, `y3` and `y4` stored for reference purposes. |
| `limit` | Same as argument stored for reference purposes. |

## Note

The returned values are saved by default as `LCT-mean.RData` in folder `my_path`. Additionally, single items are saved by default as TEXT files: `mat.txt`, `myparam.txt` (containing `scrng` and `limit` for reference purposes).

## Author(s)

Manfred Beckmann `<meb@aber.ac.uk>`

## See Also

`fiems_ltq_main`

## Examples

```
## Example profiles can be downloaded on the FIEmspro webpage
## 021016Pot-24_LCT_ESI_-.zip must be extracted in folder that defines 'my_path'

## For e.g.
## Not run: my_path <- "D:/Temp/021016Pot-24_LCT_ESI_-"
## The same folder should also contain a 'runinfo' file
## For e.g.
## Not run: runinfo <- "runinfo.csv"
## Process each profile defined in 'runinfo'
## Not run: tmp <- fiems_lct_main(my_path,runinfo,15, 25,50,60,limit=0.82,
              save.file=TRUE,file.name="LCT-mean.RData")
## End(Not run)

## ================================================================
## Arguments and matrices are saved in 'my_path', ideally the working
## directory of the experiment. For explanations regarding input
## arguments see below ...

# required is a file named by default 'runinfo.csv'
# (comma separated variables, generated in e.g. MS-Excel);
# the structure should be the following to ease data pre-processing:

#    A     |                  B                  |       C         |  D
# injorder | pathcdf                             | filecdf         |batch
#-------------------------------------------------------------------------
#  1       | D:/../Pot-LCT-2001-bc/Test_LCT_ESI_- | 021016MAN10.CDF | 1
#  2       | D:/../Pot-LCT-2001-bc/Test_LCT_ESI_- | 021016MAN11.CDF | 1
# and so on...
```

```
#  Columns:
# 'injorder' is injection order of samples (good for investigating drifts)
# 'pathcdf'  is path of folder containing "*.cdf"-files. Each run-sequence
#            or batch of runs might have its own folder.
# 'filecdf'  is the actual filename of an "*.cdf"-file.
# 'batch'    is the number of the batch the run belongs to (good for
#            investigating batch effects)

# In practice the file will contain further information regarding sample name,
# class/group information and probably other meta-data describing a sample.


## LCT Instrument Method for Flow-Injection-ESI-MS (FIE-MS):
## - 1 Function only, either positive or negative ionization mode
## - m/z range: 65.0-1000.0  (default max = 2000)   resolution: 4000
## - 2 min Acquisition

## Infusion Profile (Sketch):
##          _
##         / \
##        /   \
##       /     _
##      /        __
## ____/           _____________________________
## 0         0.5          1          1.5          2 [min]
##       |------|              |------|
##      [y1]   [y2]           [y3]   [y4]   [scan reading]
##        sample              background
## Using the above given LCT Instrument Method for FIE-MS
##   the actual scan readings y1 to y4 are used directly:
##       scrange = c(y1,y2,y3,y4)
##       with (ideally):  y2 - y1 = y4 - y3

## Raw data conversion to ANDI NetCDF-file:
##       DataBridge program (Dbridge, MassLynx, Micromass)
```

---

`fiems_ltq_main` *LTQ Mass Binning*

---

## Description

Main Routine for 'Mass Binning' to nominal mass and 'Mass Spectrum' generation in high-throughput Flow Injection Electrospray Ionisation Mass Spectrometry (FIE-MS). This routine reads ANDI NetCDF files (`*.cdf`) of LTQ `*.raw` data files converted in the Xconvert program (Xcalibur, Thermo Finnigan).

## Usage

```
fiems_ltq_main(my_path,runinfo, y1,y2,y3,y4,limit=0.7,
               itp=65537,itn=131073,hrng=50,lrng=15,
               save.file=TRUE,file.name="LTQ-mean.RData")
```

## Arguments

| | |
|---|---|
| my_path | A character string indicating the working directory where runinfo.csv file and folder containing *.cdf-files are located. |
| runinfo | A *.csv file containing at least the following run information (header row): pathcdf and filecdf . For details, see the description in Examples below. |
| y1 | A numeric value used for mass spectrum generation: start scan 'sample'. For details, see the description in Examples below. |
| y2 | A numeric value used for mass spectrum generation: end scan 'sample'. For details, see the description in Examples below. |
| y3 | A numeric value used for mass spectrum generation: start scan 'background'. For details, see the description in Examples below. |
| y4 | A numeric value used for mass spectrum generation: end scan 'background'. For details, see the description in Examples below. |
| limit | A numeric value defining the rounding limit for binning m/z-values to nominal mass. |
| itp | itp is an identifier for Scan Type 'full' and Data Type 'centroid' (Instrument Setup, Xcalibur) for data scans acquired in positive ionisation mode using the LTQ (Default=65537). |
| itn | itp is an identifier for Scan Type 'full' and Data Type 'centroid' (Instrument Setup, Xcalibur) for data scans acquired in negative ionisation mode using the LTQ (Default=131073). |
| hrng | hrng is an identifier for the high mass range. The argument has to match the 'First Mass (m/z)' of the scan range used for acquiring data using the LTQ 'Instrument Setup -> Mass Range: normal'. |
| lrng | lrng is an identifier for the low mass range. The argument has to match the 'First Mass (m/z)' of the scan range used for acquiring data in LTQ 'Instrument Setup -> Mass Range: low'. |
| save.file | A logical value indicating whether or not to save the results (default is TRUE). |
| file.name | A character for saved file name if save.file is TRUE. |

## Details

This routine is designed to handle four Scan Events (Xcalibur, Thermo Finnigan). Each *.cdf-file will result four mass spectra. Principle in brief: load *.cdf-file (pathcdf and filecdf information in runinfo.csv); sort scans (m/z and intensity values) into four lists. For each list: bin m/z-values to nominal mass between 'limit - 1' and 'limit'; sum up intensities of binned m/z values; generate sample matrix 'smat' between scans y1 and y2 and background matrix 'bmat' between scans y3 and y4; subtract: mat=smat-bmat; calculate mean of resulting matrix 'mat'; potential negative values are set to 'zero'. The implemented timer-function should be accurate for up to 24 hours which could comprise more than 7000 *.cdf-files per experiment. See the description in the Examples below.

## Value

A list containing the following components:

| | |
|---|---|
| posh | Matrix [runs x nominal masses] of high mass range in positive ionisation mode. |
| posl | Matrix [runs x nominal masses] of low mass range in positive ionisation mode. |
| negh | Matrix [runs x nominal masses] of high mass range in negative ionisation mode. |

| | |
|---|---|
| negl | Matrix [runs x nominal masses] of low mass range in negative ionisation mode. |
| runinfo | Same as argument stored for reference purposes. Additional information for each run like sample name or class can be used for further analysis (e.g. nlda). |
| scrng | A vector of y1, y2, y3 and y4 stored for reference purposes. |
| limit | Same as argument stored for reference purposes. |

## Note

The returned values are saved by default as LTQ-mean.RData in folder my_path. Additionally, single items are saved by default as TEXT files: posh.txt, posl.txt, negh.txt, negl.txt, myparam.txt (containing scrng and limit for reference purposes).

## Author(s)

Manfred Beckmann <meb@aber.ac.uk>

## See Also

fiems_lct_main

## Examples

```
## To run fiems_lqt_main, copy and paste the following code segment. Uncomment
## and change the file path and name appropriately.


## Example profiles can be downloading on the FIEmspro webpage
## 050509-Abr1.zip must be extrated in folder that defines 'my_path'

## For e.g.
## Not run: my_path <- "D:/Temp/050509-Abr1"
## The same folder should also contain a 'runinfo' file
## e.g.
## Not run: runinfo <- "runinfo.csv"
## Process each profile defined in 'runinfo'
## Not run: tmp <- fiems_lct_main(my_path,runinfo,35,95,190,250,limit=0.82,
               save.file=TRUE,file.name="LTQ-mean.RData")
## End(Not run)

## ================================================================
## Arguments and matrices are saved in 'my_path', ideally the working
## directory of the experiment. For explanations regarding input
## arguments see below ...

## required is a file named by default 'runinfo.csv'
## (comma separated variables, generated in e.g. MS-Excel)
## the structure should be the following to ease data pre-processing:

##    A     |             B                    |    C       |   D
## injorder | pathcdf                          | filecdf    | batch
##----------------------------------------------------------------
##  1       | D:/../070122-ABR1-A-repeat/cdf   | 01.cdf     | 1
##  2       | D:/../070122-ABR1-A-repeat/cdf   | 02.cdf     | 1
## and so on...
```

```
##  Columns:
## 'injorder' is injection order of samples (good for investigating drifts)
## 'pathcdf'  is path of folder containing "*.cdf"-files. Each run-sequence
##            or batch of runs might have its own folder.
## 'filecdf'  is the actual filename of an "*.cdf"-file.
## 'batch'    is the number of the batch the run belongs to (good for
##            investigating batch effects)

## In practice the file will contain further information regarding sample name,
## class/group information and probably other meta-data describing a
## sample.

## LTQ Instrument Method for Flow-Injection-ESI-MS (FIE-MS):
## - 1 Segment, 5 min Acquisition
## - 4 Scan Events:
##   -- 1: ITMS + c norm o(50.0-2000.0)
##   -- 2: ITMS + c low injrf=20.0 o(15.0-200.0)
##   -- 3: ITMS - c norm o(50.0-2000.0)
##   -- 4: ITMS - c low injrf=20.0 o(15.0-200.0)

## Infusion Profile (Sketch):
##           _
##          / \
##         /   \
##        /     _
##       /        ______
## _____/                 ____________________________
## 0       1        2        3         4         5 [min]
##       |--- ---|              |-------|
##      [x1]   [x2]            [x3]    [x4]   [scan reading]
##        sample                background
## Using the above LTQ Instrument Method for FIE-MS
##   the actual scan readings x1 to x4 of e.g. scan event 1 have to be
##   subtracted by 1 (the Scan Event) and
##   divided by 4 (total of 4 Scan Events):
##      e.g.  [y1] = ([x1]-1)/4  =>  scrange = c(y1,y2,y3,y4)
##      with (ideally):  y2 - y1 = y4 - y3

## Raw data conversion to ANDI NetCDF-file:
##      XConvert-program (Xcalibur, Thermo-Finnigan)
```

---

`fs.mrpval` *Significance of Feature Ranking*

---

## Description

Computation of the pseudo mrp-value from a resampling based feature ranking strategy. `qtl` represents the fraction of presumedly informative features. The decision is based on the average rank across all resampling steps. 1-`qtl` represents the fraction of features that serves to estimate the null distribution of ranks (i.e. ranks of uninformative variables).

## Usage

```
fs.mrpval(x,qtl=0.75)
```

## Arguments

| | |
|---|---|
| x | A list returned from feat.rank.re. |
| qtl | A numeric value of probability with values in [0,1]. |

## Value

A list with components:

| | |
|---|---|
| stats | Original feature ranking statistics. |
| fs.rank | Feature ranking vector. |
| fs.order | Feature order vector. |
| sdrank | Feature rank standard deviation. |
| mrpval | Individual feature mrp-value. |
| Ug | Uninformative variables. |
| nnull | Total number of uninformative variables. |
| qtl | Quantile qtl used. |

## Author(s)

David Enot <dle@aber.ac.uk> and Wanchang Lin <wll@aber.ac.uk>

## References

Zhang, C., Lu,X. and Zhang, X. (2006). Significance of Gene Ranking for Classification of Microarray Samples. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, VOL. 3, NO. 3, pp. 312-320.

## See Also

feat.rank.re, fs.summary

## Examples

```
## load abr1
data(abr1)
y   <- factor(abr1$fact$class)
x <- preproc(abr1$pos , y=y, method=c("log10","TICnorm"),add=1)[,110:500]
## Select classes 1 and 2
dat <- dat.sel(x, y, choices=c("1","2"))
x <- dat$dat[[1]]
y <- dat$cl[[1]]

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

## multiple rankings using AUC
z      <- feat.rank.re(x,y,method="fs.auc",pars = pars,tr.idx=tr.idx)
```

```
## Compute stability mr-p value using the 25% worst features as irrelevant
res <- fs.mrpval(z,qtl=0.75)

## print content of res
names(res)

## list of features to form the null distribution of ranks
print(res$Ug)
```

---

`fs.summary` *Feature Ranking Resampling Summary*

---

## Description

Wrapper that aggregates results obtained from a feature ranking resampling strategy. If p values have been calculated by the feature ranking technique on the overall dataset, adjusted p-values by one of the methods available in `p.adjust` are also returned.

## Usage

```
fs.summary(res1,res2,padjust="fdr",sorting=TRUE)
```

## Arguments

| | |
|---|---|
| `res1` | A list returned from `feat.rank.re`. |
| `res2` | A list returned from `fs.mrpval`. |
| `padjust` | One of the methods in `p.adjust`. |
| `sorting` | Should the results be sorted according to the feature ordering calculated on the overall data? |

## Value

A matrix of statistics. For details, see `Note` below.

## Note

The output matrix with number of rows corresponding to the number of variables and number of columns to:

- Feature ranking quantity computed on the whole dataset
- Feature rank in decreasing order of saliency
- p-value if available with feature ranking technique (optional)
- adjusted p-value if p-value available (optional)
- average feature rank across every resampling steps
- standard deviation of the feature rank across every resampling steps
- pseudo p-value calculated from the resampling strategy

### Author(s)

David Enot and Wanchang Lin <dle, wll@aber.ac.uk>.

### See Also

feat.rank.re, fs.mrpval, p.adjust

### Examples

```
## load abr1
data(abr1)
y   <- factor(abr1$fact$class)
x <- preproc(abr1$pos , y=y, method=c("log10","TICnorm"),add=1)[,110:500]
## Select classes 1 and 2
dat <- dat.sel(x, y, choices=c("1","2"))
x <- dat$dat[[1]]
y <- dat$cl[[1]]

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

## multiple rankings using AUC
z      <- feat.rank.re(x,y,method="fs.auc",pars = pars,tr.idx=tr.idx)

## Compute stability mr-p value using the 75% worst features as irrelevant
res <- fs.mrpval(z,qtl=0.25)

## print content of res
names(res)

res.1 <- fs.summary(z, res, sorting=TRUE)

## Print the 10 best features
print(res.1[1:10,])

##### Example of output with a feature ranking technique that returns p-value
z      <- feat.rank.re(x,y,method="fs.welch",pars = pars,tr.idx=tr.idx)
res <- fs.mrpval(z,qtl=0.25)
names(res)
## p-value correction with fdr
res.1 <- fs.summary(z, res, padjust = "fdr", sorting=TRUE)
## Print the 10 best features
res.1[1:10,]
```

---

## fs.techniques *Implementation of Feature Ranking Techniques*

---

### Description

Implementation of feature ranking techniques.

## Usage

```
fs.anova(x,y,...)
fs.auc(x,y)
fs.bw(x,y)
fs.kruskal(x,y,...)
fs.mi(x,y)
fs.relief(x,y)
fs.rf(x,y,...)
fs.snr(x,y)
fs.welch(x,y,...)
```

## Arguments

| | |
|---|---|
| x | A data frame or matrix of data set. |
| y | A factor or vector of class. |
| ... | Optional arguments to be passed to the feature ranking method. |

## Details

Several techniques are implemented in the current packages:

**fs.anova:** Wrapper for function `oneway.test`. Performs an analysis of variance to test whether means from normal distributions are identical. It assumes that group variances are not necessarily equal. The F value is used to compute feature ranks - Two and multiple class problems both allowed.

**fs.auc:** Compute the area under the simple ROC curve (x axis: false positive, y-axis: true positive rate) for each individual feature. The actual value of the AUC (if class 1 > class 2) or its complement (if class 1 < class 2) is used to get the feature ranking - Two class problems only.

**fs.bw:** Compute the ratio of between-group to within-group sums of squares for each feature without assuming any particular data distributions - Two and multiple class problems both allowed.

**fs.kruskal:** Wrapper for function `kruskal.test` - Non parametric alternative that handles two and multiple class problems.

**fs.mi:** Compute the mutual information between the two classes - Two class problems only.

**fs.relief:** Implementation of the RELIEF algorithm to calculate relevance scores in a multivariate fashion - Two and multiple class problems both allowed.

**fs.rf:** Wrapper for randomForest function to compute importance scores in a multivariate fashion. The mean decrease in accuracy is used to calculate feature scores. Further arguments related to the random forests algorithm can also be passed - Two and multiple class problems both allowed.

**fs.snr:** Compute the signal to noise ratio for each feature. The absolute value of the SNR is reported and used for accessing feature ranks - Two class problems only.

**fs.welch:** Performs a univariate t-test to test whether group means from normal distributions are identical assuming that group variances may not be necessarily equal. The absolute value of the t-test statistics is returned and used to compute feature ranks - Two classes problems only.

## Value

A list with components:

| | |
|---|---|
| fs.rank | A vector of feature ranks. |

| | |
|---|---|
| fs.order | A vector of feature ids in decreasing order of saliency. |
| stats | A vector of the original statistic/quantity describing feature saliency. |
| pval | A vector of p values if calculated by the feature ranking method. |

## Author(s)

David Enot <dle@aber.ac.uk> and Wanchang Lin <wll@aber.ac.uk>.

## References

Dudoit, S., Fridlyand, J. and Speed, T.P. (2002). Comparison of discrimination methods for classification of tumors using gene expression data. *Journal of the American Statistical Association*. Vol.97, No.457, 77-87.

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.

Jeffery, I. B., Higgins,D. G. and Culhane,A. C. (2006). Comparison and evaluation of methods for generating differentially expressed gene lists from microarray data. *BMC Bioinformatics*, 7:359.

Chen, D.,Liu, Z., Ma, X. and Hua,D. (2005). Selecting Genes by Test Statistics. *Journal of Biomedicine and Biotechnology*. 2005:2, 132 - 138.

Golub, T. R., et al., (1999). Molecular classification of cancer: class discovery and class prediction by gene expression monitoring. *Science*, 286:531-537.

## See Also

oneway.test, kruskal.test, randomForest, feat.rank.re.

## Examples

```
## prepare data set
data(abr1)
y   <- factor(abr1$fact$class)
x <- preproc(abr1$pos , y=y, method=c("log10","TICnorm"),add=1)[,110:500]
## Only test for class 1 and 2
dat <- dat.sel(x, y, choices=c("1","2"))
mat <- dat$dat[[1]]
cl <- dat$cl[[1]]

## apply SNR method for feature ranking
res <- fs.snr(mat,cl)
names(res)


## Template R function for a user defined feature ranking function,
## which can be used in re-sampling based feature selection
## function: feat.rank.re.
fs.custom <- function(x, y)
{
### -------- user defined feature selection method goes here ----------
## As an example, generate random importance score
  stats        <- abs(rnorm(ncol(x)))
  names(stats) <- names(x)
```

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

  ### Generate rank and order
  ### Here the importance score is in decreasing order
  fs.rank <- rank(-stats, na.last = TRUE, ties.method = "random")
  fs.order <- order(fs.rank, na.last = TRUE)
  names(fs.rank) <- names(stats)
  nam <- names(stats[fs.order])
  ### return results
  list(fs.rank = fs.rank, fs.order = fs.order, stats = stats)
}

## apply fs.custom for feature ranking
res <- fs.custom(mat,cl)
names(res)
```

---

`ftrank.agg` *Aggregation of resampling based feature ranking results*

---

## Description

Aggregate `feat.rank.re` objects and list of `feat.rank.re` objects to form `ftrank.agg` object. The main utilities of this function is to concatenate in a single list various results derived from several `feat.rank.re` calls in order to facilitate post analysis additional treatments and sorting of the results.

## Usage

```
ftrank.agg(...)
```

## Arguments

| | |
|---|---|
| `...` | `feat.rank.re` objects and/or list of `feat.rank.re` objects |

## Details

`ftdef` filed in the result list is a table with 9 columns which are automatically generated to summarise the content of each individual `feat.rank.re`. it is also aimed at avoiding confusions if the same method is applied on the same discrimination problem but with different settings, different resampling partitioning or even different data sets. Each column is described as follows:

**Mod:** Unique identifier for each resampling based feature rankings.

**Alg:** Name of the classification technique as specified in the call of `feat.rank.re`.

**Arg:** Arguments passed to the FR technique during the call of `feat.rank.re`.

**Pars:** Summary of the resampling strategy adopted during the call of `feat.rank.re`.

**Dis:** Discrimination task involved. By default, this is equal to the actual levels of the class vector passed to `feat.rank.re` separated by ~.

**AlgId:** Unique algorithm identifier based on the columns Alg, Arg and Pars so that no confusion is possible with Alg if several rankings have been built with the same FR technique but with different parameters and/or resampling strategy. This column can be modified by the user.

**DisId:** Unique algorithm identifier based on the columns Dis in order to simplified the name of the discrimination task if there are many classes involved and/or class level have a long name. This column can be modified by the user.

**Other:** Empty column that can be amended to store extra information.

## Value

`ftrank.agg` objects:

| | |
|---|---|
| `ftrank` | List of `feat.rank.re` objects |
| `frdef` | Summary of each `feat.rank.re` object - See details |

## Author(s)

David Enot <dle@aber.ac.uk>

## See Also

`feat.rank.re`

## Examples

```
data(abr1)
y   <- factor(abr1$fact$class)
x <- preproc(abr1$pos , y=y, method=c("log10","TICnorm"),add=1)[,110:500]
dat <- dat.sel1(x, y, pwise=list(c("1","2"),c("3","2")),mclass=NULL,
  pars=valipars(sampling="boot",niter=2,nreps=5))

resauc = lapply(dat, function(x) feat.rank.re(x,method="fs.auc"))
resrf = lapply(dat, function(x) feat.rank.re(x,method="fs.rf",ntree=100))

mfr=ftrank.agg(resauc,resrf)

### Print out characteristics of each individual FR objects
print(mfr$frdef)

### Number of objects in mfr
length(mfr$ftrank)

### This is FR object num.1
mfr$ftrank[[1]]
```

---

# grpplot *Scatter Plot by Group*

---

## Description

Create a scatter plot by group.

## Usage

```
grpplot(dat, group, col = NULL, pch = NULL, cex = 1,
        legend.loc = "rightside",...)
```

### Arguments

| | |
|---|---|
| dat | A matrix or data frame to be plotted. |
| group | A factor or vector giving group information of columns of dat. |
| col | A strings of colors recognized by the plot function. Or a vector of color values. For more details, see ?colors. |
| pch | A vector of symbol values recognized by the plot function. |
| cex | Size of symbols. |
| legend.loc | Position of legend. The location should be one of bottomright, bottom, bottomleft, left, topleft, top, topright, right, center,mousepoint and rightside. |
| ... | Additional graphics parameters passed to plot, such as main, xlab and ylab. |

### Details

The scatter plot is by columns of dat. If legend.loc is mousepoint, legend will be located by the mouse point.

### Author(s)

Wanchang Lin <wll@aber.ac.uk>

### Examples

```
data(iris)
grpplot(iris[,1:2], iris[,5],main="IRIS DATA",legend.loc="topleft")

## color values for the group and symbol size
grpplot(iris[,1:2], iris[,5],main="IRIS DATA",col=c(4,5,6),cex=1.2,
        legend.loc="topright")

## color and symbols for the group. legend location
grpplot(iris[,c(1,3)], iris[,5], col=c("red", "green3", "blue"),
        pch=c(15,16,17), main="IRIS DATA",legend.loc="topleft")

## plot vector
grpplot(iris[,3], iris[,5],main="IRIS DATA",legend.loc="top")

## pairs plot.
grpplot(iris, iris[,5],main="IRIS DATA")
```

---

## hca.nlda *Hierarchical Clustering for Class 'nlda'*

---

### Description

Hierarchical clustering based on the Mahalanobis distances between group centers for class nlda. Group centers coordinates of the training data points can be calculated using all possible discriminant functions or a selected number of discriminant functions.

### Usage

```
hca.nlda(x, method="complete",df2use=NULL,
         main="Aggregation of group centres",
         ylab="Mahalanobis distance",
         xlab="", sub="",...)
```

### Arguments

| | |
|---|---|
| x | An object of class nlda. |
| method | Agglomeration method to be used. This should be an unambiguous abbreviation of "ward", "single", "complete", "average", "mcquitty", "median" or "centroid". |
| df2use | Discriminant functions to be included in the HCA (by default all of DFs are considered if df2use=NULL). |
| main, sub, xlab, ylab | Character strings for annotating the HCA plot. For details, see plclust. |
| ... | Additional arguments to plclust. For details, see plclust. |

### Author(s)

David Enot <dle@aber.ac.uk> and Wanchang Lin <wll@aber.ac.uk>.

### See Also

nlda, predict.nlda

### Examples

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

## build nlda model
model    <- nlda(dat,cl)

## HCA using all DFs
hca.nlda(model)

## or only using the first 2 DFs
hca.nlda(model,df2use=1:2)
```

---

koptimp *Imputation of Low Values*

---

## Description

Wrapper function to select an optimal number of neighbours (k) in impute.knn from the IMPUTE package. For several values of k, predictions made on random data points by impute.knn are compared to their original value to calculate the root mean squared error. In the original matrix, thres corresponds to the limit under which intensities are considered missing. perc represents the percentage of "non missing" intensities randomly selected to estimate RMSE. The optimal number koptim corresponds to number of k that improves RMSE by less than 10%. This value is automatically used for computing the resulting matrix x matrix.

## Usage

```
koptimp(x,thres=1,log.t=TRUE,lk=3:10,perc=0.1,niter=10,...)
```

## Arguments

| | |
|---|---|
| x | A data frame or matrix to be imputed. |
| thres | Threshold below which intensities in x are considered missing. |
| log.t | A logical which specifies whether or not the log transformation is performed on the data set before imputation. |
| lk | A vector of numbers of neighbours to be tested. |
| perc | Percentage of non-low value to be randomly selected. |
| niter | Number of iteration. |
| ... | Arguments passed to or from other methods. |

## Value

A list containing the following components:

| | |
|---|---|
| x | An imputed data matrix using k=koptim. |
| koptim | Optimal number of neighbors found in lk. |
| rmse | Root mean squared error matrix (niter by length of lk). |

## Note

Version of package impute must be 1.8.0 or greater. At the moment of the package writing, only the package available on the Bioconductor website seemed to be regularly updated

## Author(s)

David Enot <dle@aber.ac.uk>

## References

Hastie, T., Tibshirani, R., Sherlock, G., Eisen, M., Brown, P. and Botstein, D.(1999). Imputing Missing Data for Gene Expression Arrays, *Stanford University Statistics Department Technical report*. http://www-stat.stanford.edu/~hastie/Papers/missing.pdf

Olga Troyanskaya, Michael Cantor, Gavin Sherlock, Pat Brown, Trevor Hastie, Robert Tibshirani, David Botstein and Russ B. Altman, (2001). Missing value estimation methods for DNA microarrays. *Bioinformatics*. Vol. 17, no. 6, Pages 520-525.

### Examples

```
## load data
data(abr1)
mat <- abr1$pos[,110:300]

## find an optimal number of k between 3 and 6 to impute values lower than 1
## 10 perc. of intensities >1 are used to evaluate each solution
## imputation is done with the log transformed matrix
res <- koptimp(mat,thres=1,log.t=TRUE,lk=3:6,perc=0.1,niter=5)
names(res)

## check RMSE of the solutions at various k
boxplot(res$rmse,xlab="Number of neighbours",ylab="Root mean square error")

## Do the imputation with a given k
## thres=1 and log.t=TRUE
mat[mat <= 1] <- NA ; mat <- log(mat)
## uses k=6 for example
mimp <- t(impute.knn(t(mat), k = 6, 1, 1, maxp = ncol(mat))$data)
## transform to the original space
mimp <- exp(mimp)
```

---

## mc.agg *Aggregation of classification results*

---

### Description

Aggregate `accest` objects and list of `accest` objects to form `mc.agg` object. The main utilities of this function is to concatenate in a single list various results derived from several `accest` calls in order to facilitate post analysis additional treatments as well as exporting the results.

### Usage

```
mc.agg(...)

## Default S3 method:
mc.agg(...)
```

### Arguments

| | |
|---|---|
| `...` | accest objects and or list of accest objects |

### Details

The length of the resulting list is equal to the total number of `accest` objects (i.e one resampling experiment) plus one field that summarises each `accest`. The later is a table with 8 columns which are automatically generated to and also avoid confusions if the same method is applied on the same discrimination problem but with different settings, different resampling partitioning or even different data sets. Note that columns 1 to 5 are automatically generated from the output of `accest` where as columns 6 is based on columns 2-4. Each column is described as follows:

**Mod:** Unique identifier for each resampling based feature rankings.

**Alg:** Name of the classification technique as specified in the call of accest.

**Arg:** Arguments passed to the classifier during the call of accest.

**Pars:** Summary of the resampling strategy adopted during the call of accest.

**Dis:** Discrimination task involved. By default, this is equal to the actual levels of the class vector passed to accest separated by ~.

**AlgId:** Unique algorithm identifier based on the columns Alg, Arg and Pars so that no confusion is possible with Alg column if several classification have been performed with the same discrimination technique but with different parameters and/or resampling strategy. This column can be modified by the user.

**DisId:** Unique algorithm identifier based on the columns Dis in order to simplified the name of the discrimination task if there are many classes involved and/or class level have a long name. This column can be modified by the user.

**Other:** Empty column that can be amended to store extra information.

## Value

mc.agg objects:

| | |
|---|---|
| clas | List of accest objects |
| cldef | Summary of each accest object - See details |

## Author(s)

David Enot <dle@aber.ac.uk>

## See Also

accest,mc.summary, mc.comp.1, mc.roc

## Examples

```
data(iris)
dat=as.matrix(iris[,1:4])
cl=as.factor(iris[,5])
lrnd=sample(1:150)[1:50]
cl[lrnd]=sample(cl[lrnd])
pars   <- valipars(sampling = "boot",niter = 2, nreps=10)
dat1=dat.sel1(dat,cl,pwise=list(),mclass=list(),pars=pars)

res1=accest(dat1[[1]],clmeth="randomForest",ntree=100,seed=1)
res2=lapply(dat1,function(x) accest(x,clmeth="lda"))

mc=mc.agg(res1,res2)
###Print the content
mc

### Classification task num. 1:
mc$clas[[1]]

## As other functions are using the column 5 and 6 to sort and print the results,
## you can replace them by something more informative
## for e.g. Alg and Dis
mc$cldef[,6]<-mc$cldef[,2]
```

```
mc$cldef[,7]<-c("set~vir","set~vir","set~vir","ver~vir","ver~vir")
mc$cldef
```

---

`mc.comp.1` *Multiple Classifier Predictions Comparison*

---

## Description

Function to test for significant differences between predictions made by various classifiers (method and/or settings) built on the same partitioning schema. This function requires that explicit class predictions for each fold and iteration are contained in the classifier object of the type of `accest`. For each pairwise comparison: mean of the differences, variance associated, student t-statistics and corresponding p value are reported in a table. Subsequent multiple testing correction is applied if more than two classifiers are involved. Note that column **DisId** is used to sort the classifiers according to the discrimination task and **DisId** and **AlgId** will be used to report the results. Of course, it is also assumed that partitioning for models built with two different classifiers is identical.

## Usage

```
mc.comp.1(mc.obj,lmod=NULL,p.adjust.method="holm")

## Default S3 method:
mc.comp.1(mc.obj,lmod=NULL,p.adjust.method="holm")
```

## Arguments

| | |
|---|---|
| `mc.obj` | `mc.agg` object - See details `mc.agg` |
| `lmod` | List of models to be considered - Default: all of them |
| `p.adjust.method` | Multiple testing correction. See details in `p.adjust` |

## Value

`mc.comp.1` object:

| | |
|---|---|
| `res` | Summary of classifier pairwise comparisons for each discrimination task |
| `cltask` | Discrimination task(s). |
| `title` | Title for printing function. |

## Note

See publications mentioned below.

## Author(s)

David Enot <dle@aber.ac.uk>

## References

Berrar, D., Bradbury, I. and Dubitzky, W. (2006). Avoiding model selection bias in small-sample genomic datasets. *Bioinformatics*. Vol.22, No.10, 1245-125.

Bouckaert, R.R.,and Frank, E., (2004). Evaluating the Replicability of Significance Tests for Comparing Learning Algorithms. *Proc 8th Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Vol.3054, 3-12

## See Also

mc.agg

## Examples

```
data(iris)
x <- as.matrix(subset(iris, select = -Species))
y <- iris$Species
pars   <- valipars(sampling = "cv",niter = 10, nreps=5, strat=TRUE)
tr.idx <- trainind(y,pars=pars)
## RF model based one tree
acc1   <- accest(x, y, clmeth ="randomForest", pars = pars, tr.idx=tr.idx,ntree=1)
## RF model based 100 trees
acc2   <- accest(x, y, clmeth = "randomForest", pars = pars, tr.idx=tr.idx,ntree=100)
### RF model where the minimum size of terminal nodes is set to a value greater
## than the maximum number of samples per class (oups!)
acc3   <- accest(x,y, clmeth = "randomForest", pars = pars, tr.idx=tr.idx,ntree=1,nodesiz

clas=mc.agg(acc1,acc2,acc3)
res.comp<-mc.comp.1(clas,p.adjust.method="holm")

## No significant differences between 1 and 2
## Of course classifiers 1 and 2 performs significantly better than 3
## by default
res.comp

## with a few more decimals...
print(res.comp,digits=4)

## Print results in a file
## Not run: print(res.comp,digits=2,file="tmp.csv")
```

---

# mc.meas.iter *Summary of a predictor in mc.agg object*

---

## Description

Convenience function to output statistics related to accuracy, AUC or margins at each iteration for one model or a selection of models contained in a mc.agg object (see details mc.agg).

## Usage

```
mc.meas.iter(mc.obj, lmod = NULL,type="acc",nam="Model")
```

### Arguments

| | |
|---|---|
| mc.obj | mc.agg object - See details mc.agg |
| lmod | List of models to be considered - Default: all models |
| type | Predictor type - Can be either acc (accuracy), auc (AUC), mar (margin or equivalent) |
| nam | List of names to be used in the result - Names given here corresponds to the column name of mc.obj$cldef |

### Value

Data frame containing statistic of interest at each iteration.

### Author(s)

David Enot <dle@aber.ac.uk>

### See Also

mc.agg

### Examples

```
data(iris)
dat=as.matrix(iris[,1:4])
cl=as.factor(iris[,5])
lrnd=sample(1:150)[1:50]
cl[lrnd]=sample(cl[lrnd])  ## add a bit of misclassification for fun
pars   <- valipars(sampling = "cv",niter = 10, nreps=4)
dat1=dat.sel1(dat,cl,pwise="virginica",mclass=NULL,pars=pars)

res1=lapply(dat1,function(x) accest(x,clmeth="lda"))
res2=lapply(dat1,function(x) accest(x,clmeth="randomForest",ntree=50))

## Aggregate res1 and res2
mc=mc.agg(res1,res2)

## AUC in each model
auc.iter<-mc.meas.iter(mc,type="auc",nam=c("DisId","Alg"))
## Plot them
boxplot(auc.iter)
## Print on the screen
print(auc.iter)
```

---

## mc.roc *Generate ROC curves from several classifiers*

---

## Description

Convenience function to generate a ROC curve from several runs and iterations for one model or a selection of models contained in a `mc.acc` object (see details `mc.agg`). This function allows the averaging of several ROC curves produced on each data partitioning of the resampling strategy (i.e cross-validation runs repeated or not several times). Three methods are available to perform the averaging: "horiz" (horizontal), "vert" (vertical) and "thres" (thresholding). (see reference for further details). The default value ("all") means that each data points from individual ROC curves are strictly concatenated into a single ROC curve.

## Usage

```
mc.roc(mc.obj, lmod = 1, method = "all")
## Default S3 method:
mc.roc(mc.obj, lmod = 1, method = "all")
```

## Arguments

| | |
|---|---|
| `mc.obj` | `mc.agg` object - See details `mc.agg` |
| `lmod` | List of models to be considered - Default: all of them |
| `method` | Aggregation method ("all", "thres", "vert", "horiz") |

## Value

`roc.list` object is a list of two components:

| | |
|---|---|
| `roc` | List of ROC curves of length equal to the number of models. For each curve true positive (tpr), false positive rate (fpr), decision boundary threshold (thres) and type (type) of ROC aggregation are given. |
| `cldef` | Identical to `cldef` in `mc.agg`. |

## Author(s)

David Enot `<dle@aber.ac.uk>`

## References

Fawcett, T. (2004). ROC graphs: notes and practical considerations for researchers. `Technical Report HPL-2003-4`

## See Also

`plot.mc.roc`,`mc.agg`

## Examples

```
data(iris)
dat=as.matrix(iris[,1:4])
cl=as.factor(iris[,5])
lrnd=sample(1:150)[1:50]
cl[lrnd]=sample(cl[lrnd])
pars   <- valipars(sampling = "cv",niter = 2, nreps=10)
dat1=dat.sel1(dat,cl,pwise="virginica",mclass=NULL,pars=pars)

res1=lapply(dat1,function(x) accest(x,clmeth="lda"))
```

```
res2=lapply(dat1,function(x) accest(x,clmeth="randomForest",ntree=50))
mc=mc.agg(res1,res2)

roc.sv=mc.roc(mc,lmod=1:4,method="thres")
print(roc.sv)
```

---

`mc.summary` *Summary of multiple classifiers objects*

---

## Description

Convenience function to output statistics related to accuracy, AUC and margins for a selection of models. If `sortDis=TRUE`, results are grouped by discrimination task (value contained in DisId column of `mc.obj$cldef`). If `sortDis=FALSE`, results are grouped by classifier algorithm (value contained in column AlgId of `mc.obj$cldef`).

## Usage

```
mc.summary(mc.obj,lmod=NULL,sortDis=TRUE)

## Default S3 method:
mc.summary(mc.obj, lmod = NULL, sortDis = TRUE)
```

## Arguments

| | |
|---|---|
| `mc.obj` | `mc.agg` object |
| `lmod` | List of models to be considered - Default: all of them |
| `sortDis` | Should the results be sorted by discrimination task? If FALSE, results are group by classifier techniques |

## Value

`mc.summary` object:

| | |
|---|---|
| `res` | List of results |
| `cltask` | Discrimination task(s) or classification algorithm(s) used. |
| `title` | Title for printing function. |

## Author(s)

David Enot <dle@aber.ac.uk>

## See Also

`mc.agg`

## Examples

```
data(iris)
dat=as.matrix(iris[,1:4])
cl=as.factor(iris[,5])
lrnd=sample(1:150)[1:50]
cl[lrnd]=sample(cl[lrnd])
pars   <- valipars(sampling = "cv",niter = 2, nreps=10)
dat1=dat.sel1(dat,cl,pwise="virginica",mclass=NULL,pars=pars)

res1=lapply(dat1,function(x) accest(x,clmeth="lda"))
res2=lapply(dat1,function(x) accest(x,clmeth="randomForest",ntree=50))

## Aggregate res1 and res2
mc=mc.agg(res1,res2)

## Sort results by discrimination task
mc.summary(mc)

## Sort results by algorithm
mc.summary(mc,sortDis=FALSE)

## See what is in
names(mc.summary(mc))

## Print results in a file
## Not run: print(mc.summary(mc,sortDis=FALSE),digits=2,file="tmp.csv")
```

---

# multibc *Multiple Metabolomics Fingerprint Baseline Correction*

---

## Description

Wrapper for function `onebc` for several samples. An additional plotting parameter for visualizing the baseline correction process is added.

## Usage

```
multibc(x,wsize=50,qtl=0.1,maxy=1000,plotting=FALSE,pause=0)
```

## Arguments

| | |
|---|---|
| `x` | A data frame or matrix to be processed. |
| `wsize` | Window size. |
| `qtl` | A numeric value of probability with values in [0,1]. |
| `maxy` | A numeric value which specifies maximal intensity to be plotted on y axis. |
| `plotting` | A logical value indicating whether or not plotting. |
| `pause` | Time interval defining pause between each baseline correction in order to visualise each individual plot during the BC process if `plotting` is `TRUE`. |

## Value

A list containing the following components:

| | |
|---|---|
| x | Sample matrix after baseline subtraction. |
| bsl | Baseline matrix. |

## Author(s)

David Enot <dle@aber.ac.uk>

## See Also

onebc

## Examples

```
## load abr1
data(abr1)
mat <- abr1$pos[1:5,110:2000]

## baseline correction
res <- multibc(mat,plotting=TRUE,pause=0.5)
```

---

# nlda *Linear Discriminant Analysis for High Dimensional Problems*

---

## Description

Linear discriminant analysis for high dimensional problems. See details for implementation.

## Usage

```
  nlda(dat, ...)
  ## Default S3 method:
nlda(dat,cl,prior=NULL,scale=FALSE,comprank = FALSE, ...)

  ## S3 method for class 'formula':
nlda(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. |

| | |
|---|---|
| prior | The prior probabilities of class membership. If unspecified, the class proportions for the training set are used. If present, the probabilities should be specified in the order of the factor levels. |
| scale | A logical value indicating whether or not PCA is scaled. |
| comprank | A computation rank. |
| ... | 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, nlda combines principal components analysis (PCA) and linear discriminant analysis (LDA) for the classification problem. Because the determination of the optimal number of principal components representative for a dataset is not trivial and the number of dimensions varies from one comparison to another introducing a bias to the estimation of the separability measure, we have opted for a 2 steps procedure proposed in Thomaz, C. E. and Gillies, D. F. (2004): the number of principal components to retain is equal to the rank of the covariance matrix (usually number of training samples minus one) and the within-class scatter matrix is replaced by a version where the less reliable eigenvalues have been replaced. In addition to the proportion of explained variance in each projection, the eigenvalue is a useful diagnostic quantity (output stats).

## Value

An object of class nlda containing the following components:

| | |
|---|---|
| stats | The statistics based on the training data. |
| Tw | The proportion of trace. |
| rankmat | The rank used for LDA. |
| means | The means of training data. |
| loadings | A matrix of the coefficients of linear discriminants. |
| x | The rotated data on discriminant variables. |
| xmeans | The group means obtained from training. |
| pred | The predicted class labels of training data. |
| cl | The observed class labels of training data. |
| prior | The prior probabilities used. |
| conf | The confusion matrix based on training data. |
| acc | The accuracy rate of training data. |
| lev | The levels of class. |
| call | The (matched) function call. |

## Note

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

## Author(s)

David Enot <dle@aber.ac.uk> and Wanchang Lin <wll@aber.ac.uk>.

## References

Venables, W. N. and Ripley, B. D. (2002) *Modern Applied Statistics with S*. Fourth edition. Springer.

Ripley, B. D. (1996) *Pattern Recognition and Neural Networks*. Cambridge University Press.

Thomaz, C. E. and Gillies, D. F. (2004) A Maximum Uncertainty LDA-based approach for Limited Sample Size problems with application to Face Recognition. *Technical Report*. Department of Computing, Imperial College London.

Yang, J. and Yang J.-Y. (2003) Why can LDA be performed in PCA transformed space? *Pattern Recognition*, vol.36, 563 - 566.

## See Also

predict.nlda, plot.nlda, hca.nlda

## Examples

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

## define random training and test datasets
idx <- sample(1:nrow(dat), round((2/3)*nrow(dat)), replace=FALSE)
train.dat  <- dat[idx,]
train.t    <- cl[idx]
test.dat   <- dat[-idx,]
test.t     <- cl[-idx]

## build nlda on the training data
model    <- nlda(train.dat,train.t)
## print summary
summary(model)

## map samples on the first 2 DFs
plot(model,dimen=c(1,2),main = "Training data",abbrev = TRUE)
## map samples on all the DFs
plot(model,main = "Training data",abbrev = TRUE)

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

onebc *Metabolomics Fingerprint Baseline Correction*

## Description

Core function to perform baseline correction on one metabolomics fingerprint.

## Usage

```
onebc(x,wsize=50,qtl=0.1,maxy=1000,plotting=TRUE,title=TRUE,
      sampid=NULL)
```

## Arguments

| | |
|---|---|
| x | A numeric vector to be processed. |
| wsize | Window size. |
| qtl | A numeric value of for lower quantile probability. |
| maxy | A numeric value specifying y axis maximal value to be plotted. |
| plotting | A logical value indicating whether or not plotting. |
| title | A logical value indicating whether or not to show plot title. |
| sampid | Sample ID to be written in the title if both plotting and title are TRUE (useful when onebc called from multibc). |

## Details

The purpose of the baseline correction is to remove undesirable effects due excess chemical noise. Possible consequences of baseline drift include the possibility that the baseline may be discriminatory and that important information may be obscured in areas affected by baseline problems. A simple consensual approach consists in fitting a monotone local minimum curve to each fingerprint. Basically, the fingerprint is divided into equally spaced m/z intervals and a local minimum intensity value is returned as the baseline estimate for this region. Finally, the whole fingerprint baseline is computed by linear interpolation based on pairs made of the centre of the interval and its corresponding local minima. Intervals (argument wsize) are in the order of 30-70 amu as a trade off between the removal of relevant chemical (small interval) or estimation bias due to use of a larger interval. Rather than using the minimum value of an interval, it is also judicious to use the value corresponding to a low quantile (argument qtl) to avoid any spurious estimates due to zeros or abnormally low signals.

## Value

A list containing the following components:

| | |
|---|---|
| x | A numeric vector of the resulting fingerprint after baseline correction. |
| bsl | A numeric vector of the baseline intensities. |

## Author(s)

David Enot <dle@aber.ac.uk> and Wanchang Lin <wll@aber.ac.uk>.

## See Also

multibc

## Examples

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

## baseline correction
res <- onebc(mat[1,110:2000], qtl=0.8, sampid="1")
```

---

`outl.det` *Detection and Display Outliers*

---

## Description

Wrapper for the detection of sample outliers by computation of Mahalanobis distances using `cov.rob` from package **MASS**. The (robust) square root distance from the center is displayed alongside a 2D mapping of the data and its confidence ellipse.

## Usage

```
outl.det(x, method = "classical", conf.level = 0.975,
            dimen=c(1,2), tol = 1e-7, plotting = TRUE)
```

## Arguments

| | |
|---|---|
| `x` | A data.frame or matrix. |
| `method` | The method to be used:<br>• `mve`. Minimum volume ellipsoid.<br>• `mcd`. Minimum covariance determinant.<br>• `classical`. Classical product-moment.<br>For details, see `cov.rob` in package **MASS**. |
| `conf.level` | The confidence level for controlling the cutoff of Mahalanobis distances. |
| `dimen` | Dimensions used to plot tolerance ellipse and the data points alongside these two dimensions. |
| `tol` | The tolerance to be used for computing Mahalanobis distances (see `cov.rob` in package **MASS**) |
| `plotting` | A logical value. If `TRUE`, The Mahalanobis distances against the index of data samples and the tolerance ellipse of the data samples are plotted. |

### Details

If the number of samples is n and number of variables in a sample is p, the data set must be n > p + 1. In this case, PCA can be used to produce fewer directions of uncorrelated dimensions that explain different dimensions in the data. Due to the inherent difficulties in defining outliers, inclusion of the first few dimensions only is almost always sufficient to compute Mahalanobis distances. However in more complex designs implicating various factors and/or multiple levels, different contributions to the overall variation modelled by PCA may be confounded in such a reduced space. In such situation, the initial dataset must be decomposed into smaller problems to relate potential outlying behaviour.

### Value

A list with components:

| | |
|---|---|
| outlier | List of outliers detected. |
| conf.level | Confidence level used. |
| mah.dist | Mahalanobis distances of each data sample. |
| cutoff | Cutoff of Mahalanobis distances for outliers detection. |

### Author(s)

Wanchang Lin <wll@aber.ac.uk> and David Enot <dle@aber.ac.uk>

### Examples

```
## load abr1
data(abr1)
y   <- factor(abr1$fact$class)
x <- preproc(abr1$pos , y=y, method=c("log10","TICnorm"),add=1)[,110:1000]
## Select classes 1 and 2
tmp <- dat.sel(x, y, choices=c("1","2"))
dat <- tmp$dat[[1]]
ind <- tmp$cl[[1]]


## dimension reduction by PCA
x   <- prcomp(dat,scale=FALSE)$x

## perform and plot outlier detection using classical Mahalanobis distance
## on the first 2 PCA dimensions
res <- outl.det(x[,c(1,2)], method="classical",dimen=c(1,2),
                    conf.level = 0.975)
```

---

## parse_freq *Output Variable Frequencies in Nested Lists*

---

### Description

Parser to illustrate a possible output of accest. The function loop other lists of list and retrieve the frequency of the variable names contained in the component labelled nam.

### Usage

```
parse_freq(mlist,nam,sorting=TRUE)
```

### Arguments

| | |
|---|---|
| mlist | List of list |
| nam | Name of the component of interest |
| sorting | Should the variable name be sorted by frequency? |

### Author(s)

David Enot <dle@aber.ac.uk>

### See Also

accest, parse_vec

### Examples

```
l1=list(list(v=1),list(v=c(1,2)),list(v=1))
l2=list(list(v=c(1,2,3)),list(v=2),list(v=c(3,4)))
res=list(l1,l2)

## show list of list
print(res)

## tidy up
parse_freq(res,nam="v")

######## Second example
## Real world problem described in accest
```

---

## parse_vec *Aggregation of Vectors in Nested Lists*

---

### Description

Parser to illustrate a possible output of accest. The function loop other lists of list and aggregate vectors (single values) contained in component labelled nam.

### Usage

```
parse_vec(mlist,nam)
```

### Arguments

| | |
|---|---|
| mlist | List of list |
| nam | Name of the component of interest |

### Author(s)

David Enot <dle@aber.ac.uk>.

### See Also

accest, parse_freq

### Examples

```
l1=list(list(v=1),list(v=2),list(v=3))
l2=list(list(v=4),list(v=5),list(v=6))
res=list(l1,l2)

## show list of list
print(res)

## tidy up
parse_vec(res,nam="v")

######## Second example
## Real world problem described in accest
```

---

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

---

### Description

Plots accuracy, margin or AUC at each iteration.

### Usage

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

### Arguments

| | |
|---|---|
| x | An object of class accest. |
| toplot | Quantity to plot (default=acc, mar, auc) |
| 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 a performance estimate against iteration index.

### Author(s)

David Enot and Wanchang Lin <dle,wll@aber.ac.uk>.

### See Also

accest, accest

### Examples

```
# Iris data
data(iris)
x <- as.matrix(subset(iris, select = -Species))
y <- iris$Species
pars <- valipars(sampling="cv", niter=10, nreps=5, strat=TRUE)

acc  <- accest(x,y, clmeth = "randomForest", pars = pars, ntree=50)
acc
## plot accuracies
plot(acc)

## plot margins
plot(acc,toplot="mar")
```

---

## plot.mc.roc *Plot multiple ROC curves*

---

### Description

Plot multiple ROC curves contained in mc.roc objects (see details in mc.roc).

### Usage

```
## S3 method for class 'mc.roc':
plot(x, leg = "Model", llty = NULL, lcol = NULL, xleg = 0.5, yleg = 0.4, ...)
```

### Arguments

| | |
|---|---|
| x | mc.roc object - See details in mc.roc |
| leg | User defined legend or select information from x$cldef |
| llty | List of symbols |
| lcol | List of colors |
| xleg | Upper left corner co-ordinate on the x-axis |
| yleg | Upper left corner co-ordinate on the y-axis |
| ... | Further arguments to be passed to lines |

### Value

NULL

### Author(s)

David Enot <dle@aber.ac.uk>

### See Also

mc.roc

### Examples

```
data(iris)
dat=as.matrix(iris[,1:4])
cl=as.factor(iris[,5])
lrnd=sample(1:150)[1:50]
cl[lrnd]=sample(cl[lrnd])
pars   <- valipars(sampling = "cv",niter = 2, nreps=10)
dat1=dat.sel1(dat,cl,pwise="virginica",mclass=NULL,pars=pars)

res1=lapply(dat1,function(x) accest(x,clmeth="lda"))
res2=lapply(dat1,function(x) accest(x,clmeth="randomForest",ntree=50))
mc=mc.agg(res1,res2)

roc.sv=mc.roc(mc,lmod=1:4)

### Default plot
plot(roc.sv)

###Improve plotting by using the names contained in the "DisId" and "Alg"
plot(roc.sv,leg=c("DisId","Alg"),llty=c(1,1,2,2),lcol=c("blue","red","blue","red"))

###Improve plotting by setting the legend
plot(roc.sv,leg=c("roc1","roc2","roc3","roc4"),llty=c(1,1,2,2),lcol=c("blue","red","blue"
```

---

## plot.nlda *Plot Method for Class 'nlda'*

---

### Description

Plots a set of data on one, two or more linear discriminants.

### Usage

```
## S3 method for class 'nlda':
plot(x, panel = panel.nlda, cex = 0.7, dimen,
     abbrev = FALSE, ...)
```

### Arguments

| | |
|---|---|
| x | An object of class nlda. |
| panel | The panel function used to plot the data. |
| cex | Graphics parameter cex for labels on plots. |
| dimen | The index of linear discriminants to be used for the plot. |

| | |
|---|---|
| abbrev | Whether the group labels are abbreviated on the plots. If abbrev > 0 this gives minlength in the call to abbreviate. |
| ... | Additional arguments to plot. |

## Details

This function is a method for the generic function plot() for class nlda. The behaviour is determined by the value of dimen. For the length of dimen is greater than 2, a pairs plot is used. For the length of dimen is equal to 2, a scatter plot is drawn. Otherwise, a set of histograms or density plots are drawn.

## Author(s)

Wanchang Lin <wll@aber.ac.uk> and David Enot <dle@aber.ac.uk>.

## See Also

nlda, predict.nlda, hca.nlda

## Examples

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

## build model on all the data available
model    <- nlda(dat,cl)

## Plot second component versus first
plot(model,dimen=c(1,2),main = "Training data",abbrev = TRUE)

## Pairwise scatterplots of several components
plot(model,main = "Training data",abbrev = TRUE)
```

---

# predict.nlda *Classify Multivariate Observations by 'nlda'*

---

## Description

Classify multivariate observations in conjunction with nlda, and also project data onto the linear discriminants.

## Usage

```
## S3 method for class 'nlda':
predict(object, newdata, dim2use = NULL, ...)
```

### Arguments

| | |
|---|---|
| object | Object of class nlda. |
| newdata | A matrix or data frame of cases to be classified. |
| dim2use | The dimension of rotated data set to be used in prediction. |
| ... | Arguments passed to or from other methods. |

### Details

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

### Value

A list with components:

| | |
|---|---|
| class | The predicted class (a factor). |
| x | The projections of test data on discriminant variables. |
| prob | The posterior probabilities for the predicted classes. |
| xmeans | The group means obtained from training. |
| dim2use | The dimension of rotated data set to be used in prediction. |

### Author(s)

David Enot <dle@aber.ac.uk> and Wanchang Lin <wll@aber.ac.uk>.

### See Also

nlda, plot.nlda

### 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)

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

## apply NLDA
model    <- nlda(train.dat,train.t)
pred.te  <- predict(model, test.dat)

## confusion matrix
table(test.t,pred.te$class)
```

preproc *Data Tranformation Wrapper*

## Description

Wrapper for several techniques to perform data transformation of the original fingerprint matrix.

## Usage

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

## Arguments

| | |
|---|---|
| x | A numeric data frame or matrix to be pre-processed. |
| method | A method used to pre-process the data set. The following methods are supported:<br>• center: Centering<br>• auto: Auto scaling<br>• range: Range scaling<br>• pareto: Pareto scaling<br>• vast: Vast scaling<br>• level: Level scaling<br>• log: Log transformation (default)<br>• log10: Log 10 transformation<br>• sqrt: Square root transformation<br>• asinh: Inverse hyperbolic sine transformation<br>• TICnorm: TIC normalisation |
| add | A numeric value for addition used in the logarithmic transformations log and log10. |
| y | A factor specifying the class for each observation. It is only used by the method TICnorm. |

## Details

Purpose of normalisation is to remove inter-spectrum sources of variability that come mainly from different sample concentration, loss of sensitivity of the detector over time or degradation of certain samples. Global normalisation by rescaling each measurement within a spectrum by a constant factor, such as the sum of all the spectra intensities (TICnorm method). However, the default use of TIC normalisation can lead to the generation of spurious knowledge if the overall sample intensity is class/factor dependent. If this is the case, a straightforward approach is to remove the difference, between the average scale of the corresponding class and the average scale, from the scale factor by providing the class/factor of interest (argument y) while calling preproc.

## Value

Transformed sample matrix.

## Author(s)

Wanchang Lin <wll@aber.ac.uk> and David Enot <dle@aber.ac.uk>.

## 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[,110:2000]

## normalise data set using class dependent "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)
```

---

# summ.ftrank *Summarise multiple resampling based feature ranking outputs*

---

## Description

This routine performs further computations on a list of `feat.rank.re` objects contained in a `mfr.obj` object (see `ftrank.agg`). If only one `feat.rank.re` result from `feat.rank.re` is analyzed, it is easier to pass it first to `ftrank.agg` (see example).Two calculations are made: 1) Computation of the pseudo mrp-value from the resampling based feature (see `fs.mrpval`) and 2) adjusted p values if p values have calculated (see `p.adjust`).

## Usage

```
summ.ftrank(lclas, lmod = NULL, qtl = 0.25, padjust = "fdr")
```

## Arguments

| | |
|---|---|
| `lclas` | mfr.obj object - See details in `ftrank.agg` |
| `lmod` | List of models to be considered in lclas |
| `qtl` | Quantile - See details in `fs.mrpval` |
| `padjust` | p value adjustement method - See details in `p.adjust` |

## Details

The resulting list as two component: one is equal to the total number of `feat.rank.re` objects (i.e one resampling experiment) and one field is a table that summarises each `feat.rank.re` (as for `ftrank.agg`). In the first component, each table may have a different number of columns depending if the FR method outputs p-values or not:

**Stat:** Original statistics calculated on the overall data.

**Rank:** Original feature rank calculated on the overall data.

**pval:** Original feature p-value calculated on the overall data (optional).

**pval-xxx:** Feature adjusted p-value by method xxx if p-value available (i.e. previous column).

**mrpval-xxx:** Pseudo multiple resampling p-value using a given `qtl` value xxx.

**AvgRk:** Average feature rank calculated from the ranks found for each training data partition.

**SdevRk:** Associated feature rank standard deviation calculated from the ranks found for each training data partition.

## Value

`mfr.sum` object:

| | |
|---|---|
| `frsum` | List of tables corresponding to each `feat.rank.re` object - See details |
| `frdef` | Summary of each `feat.rank.re` object as in `ftrank.agg` |

## Author(s)

David Enot <dle@aber.ac.uk>

## See Also

`fs.mrpval`, `tidy.ftrank`, `p.adjust`

## Examples

```
data(abr1)
y   <- factor(abr1$fact$class)
x <- preproc(abr1$pos , y=y, method=c("log10","TICnorm"),add=1)[,110:500]
## Select classes 1 and 2
dat <- dat.sel1(x, y, pwise="1",mclass=NULL,pars=valipars(sampling="boot",niter=2,nreps=5

reswelch = feat.rank.re(dat[[1]],method="fs.welch")
mfr=ftrank.agg(reswelch)
print(mfr$frdef)
frsum=summ.ftrank(mfr,lmod=1,qtl=.3)

## print the FR components
print(frsum$frdef)

## have a look at the first 5 variables
print(frsum$frsum[[1]][1:5,])
```

`ticstats` *Compute and Display Total Ion Count (TIC) statistics*

## Description

By definition the total ion count of a spectrum is the sum of all the m/z signal intensities (sum over the columns). A robust regression can be built to model the effect of the injection order on the TIC of each sample. The fitting residuals are used to evaluate the median of the absolute deviations (MAD) from the linear equation. Sample TIC and linear model are then plotted according to the injection order to identify potential outliers and or structure in the data.

## Usage

```
ticstats(x,injorder=NULL,thres=3)
```

## Arguments

| | |
|---|---|
| `x` | A numeric data frame or matrix to be processed. |
| `injorder` | A numeric vector corresponding to the injection order of each sample. |
| `thres` | A numeric value of threshold for detecting outlier. |

## Details

As an easy diagnostic measure, the TIC can provide an estimation of factors that may affect the overall intensity of the run such as gradual instrument drift (e.g. resulting from loss of sensitivity of the ion source), or step changes in instrument characteristics after maintenance. Also, an examination of the TIC can reveal suspicious samples where unusually low or especially high signal intensities in some runs may be due to contamination or poorly extracted samples. A regression can be built to model the effect of the injection order on the TIC of each sample. As a conservative rule, any sample that deviates more than 2/3 (argument `thres`) times from the MAD must be examined manually to identify the origin of the different intensity behaviour and then removed before further statistical analysis if corrective measures do not improve the individual fingerprint. Further assessment of outlying samples is discussed later. In any case where a linear relationship (i.e. gradually changing TIC in sample set) is observed between the injection order and sample TIC, this dependency will be removed by TIC normalisation. If other structure related to the order of injection is noticed, for example an analytical batch effect (i.e. a step change in TIC at beginning or in middle of an injection series), the user must identify its potential origin (e.g. changes in machine calibration or mobile phase) and possibly create a new experimental factor (batch) where each step change in level corresponds to the start of a new batch.

## Value

A list containing the following components:

| | |
|---|---|
| `resid` | Sample residuals. |
| `mod` | Robust linear model. |
| `loutl` | List of outlying samples as defined by the `thres` argument. |

## Author(s)

David Enot and Wanchang Lin `<dle,wll@aber.ac.uk>`

## See Also

rlm.

## Examples

```
data(abr1)
dat <- abr1$pos
res <- ticstats(dat,injorder=NULL)
```

---

`tidy.ftrank` *Tidy up multiple resampling based ranking results.*

---

## Description

Convenience function to ease the output of `summ.ftrank` objects.

## Usage

```
tidy.ftrank(frsum, lmod = NULL, sorting = "Stat", tidy = "DisId", nam = "AlgId",
```

## Arguments

| | |
|---|---|
| `frsum` | `mfr.sum` objects |
| `lmod` | List of objects to be printed out - Default all objects |
| `sorting` | Should the results be sorted according to an argument contained in `frsum$frdef` - Default sorted by variable name |
| `tidy` | Tidy the output according to an argument contained in `frsum$frdef` |
| `nam` | Concatenate the original column names with content in to an argument contained in `frsum$frdef` |
| `decreasing` | Sorting order if argument `sorting` is not NULL |
| `file` | Write results into one file or several files |

## Details

This function has been designed to provide maximum facilities to ease both screen printing and file write of complex list of resampling based feature ranking resulting from different comparisons and/or different FR settings. It uses heavily the information contained in the FR definition table (`frdef` in the `frsum` object) to: 1) merge different tables (1 FR method on 1 dataset) in a bigger table (several FR and/or datasets) 2) sort each ranking table given a statistical output and 3) group tables into subsets corresponding to FR technique or datasets. File writing option can be rather useful when 1), 2) or 3) must be repeated manually several times:

- Parameter `nam` controls the string that can be added at the end of each column in order to avoid confusion when merging individual. For e.g. if Alg is chosen, the value contained in the column Alg of `frsum$frdef` is added to Stats, Rank etc...
- If a table should be sorted before merging, the parameter `sorting` should be set accordingly. For e.g. sorting the FR result according to the rank is easliy made by `sorting="Rank")`. The field `decreasing` should be also set to FALSE if results must presented in increasing order.

- By setting the parameter `tidy` to one of the column name of `frsum$frdef`, a list of tables involving only identical values in `tidy` will be merged together. In the example involving 5 discrimination tasks and 2 feature ranking methods illustrated below, results are either sorted by FR technique or by discrimination problem.
- Results can be redirect to a file or several files so that further manipulation can be made in a more `friendly` spreadsheet software. When the results are tidied, several files are generated.

## Value

List of tables or list of files.

## Author(s)

David Enot <dle@aber.ac.uk>

## See Also

summ.ftrank

## Examples

```
#################################################################
### Example involving 5 discrimination tasks by 2 feature ranking techniques
data(abr1)
y   <- factor(abr1$fact$class)
x <- preproc(abr1$pos , y=y, method=c("log10","TICnorm"),add=1)[,110:500]
## Select classes all pairwise problems involving class with 1
dat <- dat.sel1(x, y, pwise="1",mclass=NULL,pars=valipars(sampling="boot",niter=2,nreps=5

### Perform AUC and Random Forest ranking
resauc = lapply(dat, function(x) feat.rank.re(x,method="fs.auc"))
resrf = lapply(dat, function(x) feat.rank.re(x,method="fs.rf",ntree=100))

#################################################################
## Aggregate all the models
mfr=ftrank.agg(resauc,resrf)
print(mfr$frdef)
## Compute mrp-val for all the FR models
frsum=summ.ftrank(mfr,qtl=.7)

## print the FR components
print(frsum$frdef)

## have a look at the first 5 variables in the second model
print(frsum$frsum[[2]][1:5,])

#################################################################
##### Let's concentrate on the models 1 and 6 first
## No need to tidy here - no sorting of the results and
## add the content of "Alg" in the columns names
res=tidy.ftrank(frsum,lmod=c(1,6),tidy=NULL,sorting=NULL,nam="Alg")

## because we did not tidy the ranking tables
## rankings from models 1 and 6 are concatenated
```

```
## in the first field of res
## print out the first 5 lines
res[1:5,]

## Same as before but we sort the results according to "Stat"
## in decreasing order
res=tidy.ftrank(frsum,lmod=c(1,6),tidy=NULL,sorting="Stat",nam="Alg",decreasing=TRUE)
res[1:5,]

## Same as before but we sort the results according to "Rank"
## in increasing order (of course)
res=tidy.ftrank(frsum,lmod=c(1,6),tidy=NULL,sorting="Rank",nam="Alg",decreasing=FALSE)
res[1:5,]

#################################################################
###### Tidy all the rankings according to the discrimination task
## in this case we set tidy="Dis"
## all models: lmod=NULL
## sorting according to Stat
## append the name contained in Alg to the column name

res=tidy.ftrank(frsum,lmod=NULL,tidy="Dis",sorting="Stat",nam="Alg")

## Discrimination task tags are the name for each field of res
names(res)
## same as before for discrimination'1~2' or 1
res[[1]][1:5,]
res$'1~2'[1:5,]

## discrimination'2~3' or 2 ...
res[[2]][1:5,]


#####################################################################
###### Tidy all the rankings according to the FR method
## in this case we set tidy="Alg" for e.g.
## all models: lmod=NULL
## sorting according to Stat
## append the name contained in Dis to the column name

res=tidy.ftrank(frsum,lmod=NULL,tidy="Alg",sorting="Stat",nam="Dis")

## FR technique method are the name for each field or res
names(res)

## the top 5 variables highlighted by AUC in the 5 comparisons ...
res$'fs.auc'[1:5,]

#################################################################
###### Redirect the output to one or several files
## print each tables to a CSV files starting by "testtidy"
## followed by the string contained in tidy (here "Dis")
## Not run: tidy=tidy.ftrank(frsum,lmod=NULL,tidy="Dis",sorting="Stat",nam="Alg",file="te
```

trainind *Generation of Training Samples Indices*

## Description

Generate training samples indices. 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 indices.

## Note

To avoid any errors when using subsequent classification techniques or others, training partitions contain at least two members of each class and prediction sets at least one sample of each class. Therefore, a minimum of 3 samples per class is required. As trainind is a fairly fast, hanging during the function may come from a low number of replicate in a class and/or the choice of an inadequate value (nreps).

## Author(s)

Wanchang Lin <wll@aber.ac.uk>

## See Also

valipars, accest

## 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))
```

---

`trainind.cv` *Constrained Generation of Training Samples Indices*

---

## Description

Special case of cross-validation where the fold are defined by the user. Data are partitioned according to values contained in `cv`.

## Usage

```
trainind.cv(cv)
```

## Arguments

| | |
|---|---|
| `cv` | Vector containing the fold information |

## Value

Returns a list of training index.

## Author(s)

David Enot <dle@aber.ac.uk>

## See Also

`trainind`,`valipars`, `accest`

## Examples

```
## Load abr1
data(abr1)

## cl is the class of interest
cl  <- factor(abr1$fact$class)
dat <- preproc(abr1$pos , y=cl, method=c("log10","TICnorm"),add=1)[,110:500]

## For illustration, we use sample replicates id to form the CV folds
re  <- factor(abr1$fact$rep)

## Check representativity of cl in each fold
table(re,cl)

## Generate a trainind object using re
tr.idx <- trainind.cv(re)
pars   <- valipars(sampling="cv", niter=1, nreps=5)

## for fold num. 1
## check sample indices and replicates ids in the training set
tmp    <- tr.idx[[1]]
cl[tmp[[1]]];table(re[tmp[[1]]])     ## train idx
## and in the test set
cl[-tmp[[1]]];table(re[-tmp[[1]]])   ## test idx

##  accuracy estimation with constrained CV
acc  <- accest(dat, cl, pars = pars, clmeth = "nlda", tr.idx=tr.idx)
acc
## compare it with random CV
pars.rnd   <- valipars(sampling="cv", niter=10, nreps=5)
tr.idx.rnd <- trainind(cl,pars.rnd)
acc.rnd  <- accest(dat, cl, pars = pars.rnd, clmeth = "nlda", tr.idx=tr.idx.rnd)

## plot the histogram of the accuracies at each iteration
hist(acc.rnd$acc.iter)
```

---

`valipars` *Generate Control Parameters For Validation / Resampling*

---

## Description

Generate the control parameters for resampling or validation 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`. K-fold cross-validation (default)

- `rand`. Randomised validation (holdout)
- `boot`. Bootstrap

| | |
|---|---|
| `niter` | Number of iteration or repeat for validation. |
| `nreps` | Number of replications in each iteration (number of folds for `sampling=cv` and bootstrap for `sampling=boot`). |
| `strat` | A logical value indicating if stratification is applied to `sampling=cv, rand` and `boot`. |
| `div` | Proportion of training data randomly selected for `sampling=rand`. |

## 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 <wll@aber.ac.uk>

## See Also

`trainind`, `accest`

## 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