Package ‘mixOmics’

Type Package

Title Omics Data Integration Project

Version 6.2.0

Date 2017-08-03

Depends R (>= 2.10), MASS, lattice, ggplot2

Imports igraph, rgl, ellipse, corpcor, RColorBrewer, plyr, parallel, dplyr, tidyr, reshape2, methods

Author Kim-Anh Le Cao, Florian Rohart, Ignacio Gonzalez, Sebastien Dejean with key contributors Benoit Gautier, Francois Bartolo and contributions from Pierre Monget, Jeff Coquery, FangZou Yao, Benoit Liquet.

Maintainer Kim-Anh Le Cao <kimanh.1ecao@unimelb.edu.au>

Description Multivariate methods are well suited to large omics data sets where the number of variables (e.g. genes, proteins, metabolites) is much larger than the number of samples (patients, cells, mice). They have the appealing properties of reducing the dimension of the data by using instrumental variables (components), which are defined as combinations of all variables. Those components are then used to produce useful graphical outputs that enable better understanding of the relationships and correlation structures between the different data sets that are integrated. mixOmics offers a wide range of multivariate methods for the exploration and integration of biological datasets with a particular focus on variable selection. The package proposes several sparse multivariate models we have developed to identify the key variables that are highly correlated, and/or explain the biological outcome of interest. The data that can be analysed with mixOmics may come from high throughput sequencing technologies, such as omics data (transcriptomics, metabolomics, proteomics, metagenomics etc) but also beyond the realm of omics (e.g. spectral imaging). The methods implemented in mixOmics can also handle missing values without having to delete entire rows with missing data. A non exhaustive list of methods include variants of generalised Canonical Correlation Analysis, sparse Partial Least Squares and sparse Discriminant Analysis. Recently we implemented integrative methods to combine multiple data sets: N-integration with variants of Generalised Canonical Correlation Analysis and P-integration with variants of multi-group Partial Least Squares.

License GPL (>= 2)

URL http://www.mixomics.org
BugReports https://bitbucket.org/klecao/package-mixomics/issues
Repository CRAN
Date/Publication 2017-08-17 15:25:23 UTC
NeedsCompilation no
LazyData true

R topics documented:

auroc ................................................................. 4
background.predict ........................................... 7
block.pls ............................................................. 9
block.plsda ........................................................ 12
block.spls ........................................................ 15
block.splsda ...................................................... 19
breast.TCGA ....................................................... 23
breast.tumors ..................................................... 24
cim ................................................................. 25
cimDiablo .......................................................... 33
circosPlot .......................................................... 35
color.jet ............................................................ 38
diverse.16S ......................................................... 39
estim.regul .......................................................... 41
explained_variance ........................................... 41
get.confusion_matrix ......................................... 42
image .............................................................. 44
image.estim.regul ............................................. 45
imgCor ............................................................ 45
ipca ............................................................... 47
Koren.16S ........................................................... 49
linnerud ............................................................ 50
liver.toxicity ...................................................... 51
logratio.transfo ................................................ 52
map ............................................................... 53
mat.rank .......................................................... 54
mint.block.pls .................................................. 55
mint.block.plsda ............................................... 58
mint.block.spls ............................................... 61
mint.block.splsda ............................................ 64
mint.pca .......................................................... 67
mint.pls .......................................................... 69
mint.plsda ....................................................... 71
mint.spls ......................................................... 74
mint.splsda ...................................................... 77
mixOmics .......................................................... 80
multidrug ......................................................... 84
nearZeroVar ...................................................... 86
Description

Calculates the AUC and plots ROC for supervised objects from s/plsda, mint.s/plsda and block.plsda, block.splsda or wrapper.sgcca.

Usage

```r
## S3 method for class 'plsda'
auroc(object, newdata = object$input.X, outcome.test = as.factor(object$Y), multilevel = NULL, plot = TRUE, roc.comp = 1, ...)

## S3 method for class 'splsda'
auroc(object, newdata = object$input.X, outcome.test = as.factor(object$Y), multilevel = NULL, plot = TRUE, roc.comp = 1, ...)

## S3 method for class 'mint.plsda'
auroc(object, newdata = object$X, outcome.test = as.factor(object$Y), study.test = object$study, multilevel = NULL, plot = TRUE, roc.comp = 1, roc.study = "global", ...)

## S3 method for class 'mint.splsda'
auroc(object, newdata = object$X, outcome.test = as.factor(object$Y), study.test = object$study, multilevel = NULL, plot = TRUE, roc.comp = 1, roc.study = "global", ...)

## S3 method for class 'sgcca'
auroc(object, newdata = object$X, outcome.test = as.factor(object$Y), multilevel = NULL, plot = TRUE, roc.block = 1, roc.comp = 1, ...)
```

Arguments

- **object**: Object from one of the following supervised analysis class: "plsda", "splsda", "mint.plsda", "mint.splsda", "block.plsda" or "wrapper.sgcca"
- **newdata**: numeric matrix of predictors, by default set to the training data set (see details).
- **outcome.test**: Either a factor or a class vector for the discrete outcome, by default set to the outcome vector from the training set (see details).
- **study.test**: For MINT objects, grouping factor indicating which samples of newdata are from the same study. Overlap with object$study are allowed.
- **multilevel**: Sample information when a newdata matrix is input and when multilevel decomposition for repeated measurements is required. A numeric matrix or data frame indicating the repeated measures on each individual, i.e. the individuals ID. See examples in splsda.
plot  Whether the ROC curves should be plotted, by default set to TRUE (see details).

roc.comp  Specify the component (integer) for which the ROC will be plotted from the multivariate model, default to 1.

roc.block  Specify the block number (integer) or the name of the block (set of characters) for which the ROC will be plotted for a block.plsda or block.splsda object, default to 1.

roc.study  Specify the study for which the ROC will be plotted for a mint.plsda or mint.splsda object, default to "global".

...  external optional arguments for plotting

Details

For more than two classes in the categorical outcome Y, the AUC is calculated as one class vs. the other and the ROC curves one class vs. the others are output.

The ROC and AUC are calculated based on the predicted scores obtained from the predict function applied to the multivariate methods (predict(object)$predict). Our multivariate supervised methods already use a prediction threshold based on distances (see predict) that optimally determine class membership of the samples tested. As such AUC and ROC are not needed to estimate the performance of the model (see perf, tune that report classification error rates). We provide those outputs as complementary performance measures.

The pvalue is from a Wilcoxon test between the predicted scores between one class vs the others.

External independent data set (newdata) and outcome (outcome.test) can be input to calculate AUROC. The external data set must have the same variables as the training data set (object$X).

If newdata is not provided, AUROC is calculated from the training data set, and may result in overfitting (too optimistic results).

Note that for mint.plsda and mint.splsda objects, if roc.study is different from "global", then newdata), outcome.test and sstudy.test are not used.

Value

Depending on the type of object used, a list that contains: The AUC and Wilcoxon test pvalue for each 'one vs other' classes comparison performed, either per component (splsda, plsda, mint.plsda, mint.splsda), or per block and per component (wrapper.sgccda, block.plsda, block.splsda).

Author(s)

Benoit Gautier, Francois Bartolo, Florian Rohart

See Also

tune, perf, and http://www.mixOmics.org for more details.
Examples

```r
## example with PLSDA, 2 classes
# ------------------
data(breast.tumors)
X <- breast.tumors$gene.exp
Y <- breast.tumors$sample$treatment

plsda.breast <- plsda(X, Y, ncomp = 2)
auc plsda.breast = auroc(plsda.breast, ncomp = 1)

## example with SPLSDA
# ------------------

test.breast <- splsda(X, Y, ncomp = 2, keepX = c(25, 25))
auroc(splsda.breast, plot = FALSE)

## example with splsda with T classes
# ------------------
data(liver.toxicity)
X <- as.matrix(liver.toxicity$gene)
# Y will be transformed as a factor in the function,
# but we set it as a factor to set up the colors.
Y <- as.factor(liver.toxicity$treatment)

splsda.liver <- splsda(X, Y, ncomp = 2, keepX = c(20, 20))
auc.splsda.liver = auroc(splsda.liver, ncomp = 1)

## example with mint.plsda
# ------------------
data(stemcells)

res = mint.plsda(X = stemcells$gene, Y = stemcells$celltype, ncomp = 3,
study = stemcells$study)
auc.mint.plsda = auroc(res, plot = FALSE)

## example with mint.splsda
# ------------------
res = mint.splsda(X = stemcells$gene, Y = stemcells$celltype, ncomp = 3,
keepX = c(10, 5, 15),
study = stemcells$study)
auc.mint.splsda = auroc(res, plot = TRUE, roc.comp = 3)

## example with block.plsda
# ------------------
data(nutrimouse)
data = list(gene = nutrimouse$gene, lipid = nutrimouse$lipid)
# with this design, all blocks are connected
design = matrix(c(0,1,1,0), ncol = 2, nrow = 2,
byrow = TRUE, dimnames = list(names(data), names(data)))
```
background.predict

block.plsda.nutri = block.plsda(X = data, Y = nutrimouse$diet)
auc.block.plsda.nutri = au roc(block.plsda.nutri, block = 'lipid')

## example with block.splsda
# --------------
list.keepX = list(gene = rep(10, 2), lipid = rep(5,2))
block.splsda.nutri = block.splsda(X = data, Y = nutrimouse$diet, keepX = list.keepX)
auc.block.splsda.nutri = au roc(block.splsda.nutri, block = 1)

## End(Not run)

background.predict Calculate prediction areas

Description
Calculate prediction areas that can be used in plotIndiv to shade the background.

Usage
background.predict (object, comp.predicted = 1, dist = "max.dist",
xlim = NULL, ylim = NULL, resolution = 100)

Arguments

- object: A list of data sets (called 'blocks') measured on the same samples. Data in
  the list should be arranged in matrices, samples x variables, with samples order
  matching in all data sets.
- comp.predicted: Matrix response for a multivariate regression framework. Data should be contin-
  uous variables (see block.splsda for supervised classification and factor reponse)
- dist: distance to use to predict the class of new data, should be a subset of "centroids.dist",
  "mahalanobis.dist" or "max.dist" (see predict).
- xlim,ylim: numeric list of vectors of length 2, giving the x and y coordinates ranges for the
  simulated data. By default will be 1.2* the range of object$variates$X[,i]

resolution: A total of resolution*resolution data are simulated between xlim[1], xlim[2],
ylim[1] and ylim[2].

Details
background.predict simulates resolution*resolution points within the rectangle defined by
xlim on the x-axis and ylim on the y-axis, and then predicts the class of each point (defined by two
coordinates). The algorithm estimates the predicted area for each class, defined as the 2D surface
where all points are predicted to be of the same class. A polygon is returned and should be passed
to plotIndiv for plotting the actual background.
Note that by default xlim and ylim will create a rectangle of simulated data that will cover the plotted area of plotIndiv. However, if you use plotIndiv with ellipse=TRUE or if you set xlim and ylim, then you will need to adapt xlim and ylim in background.predict.

Also note that the white frontier that defines the predicted areas when plotting with plotIndiv can be reduced by increasing resolution.

More details about the prediction distances in ?predict and the referred publication.

Value

background.predict returns a list of coordinates to be used with polygon to draw the predicted area for each class.

Author(s)

Florian Rohart

References


See Also

plotIndiv, predict, polygon.

Examples

# Example 1
# ----------------------------------
## Not run:
data(breast.tumors)
X <- breast.tumors$gene.exp
Y <- breast.tumors$sample$treatment
splsda.breast <- splsda(X, Y, keepX=c(10, 10), ncomp=2)

# calculating background for the two first components, and the centroids distance
background = background.predict(splsda.breast, comp.predicted = 2, dist = "centroids.dist")

# default option: note the outcome color is included by default!
plotIndiv(splsda.breast, background = background)

## End(Not run)

# Example 2
# ----------------------------------
## Not run:
data(liver.toxicity)
block.pls

\[
X = \text{liver.toxicity}\$\text{gene} \\
Y = \text{as.factor(liver.toxicity}\$\text{treatment[, 4]})
\]

\[
\text{plsd}\text{a.liver} \leftarrow \text{plsd}\text{a}(X, Y, n\text{comp} = 2)
\]

# calculating background for the two first components, and the mahalanobis distance
background = background.predict(plsd\text{a.liver, comp.predicted} = 2, dist = "mahalanobis.dist")

\[
\text{plotIndiv(plsd}\text{a.liver, background = background, legend = TRUE)}
\]

## End(Not run)

---

**block.pls**

*N-integration with Projection to Latent Structures models (PLS)*

---

**Description**

Integration of multiple data sets measured on the same samples or observations, ie. N-integration. The method is partly based on Generalised Canonical Correlation Analysis.

**Usage**

\[
\text{block.pls}(X, Y, \text{indy}, n\text{comp} = 2, \text{design, scheme, mode, scale = TRUE, bias, init, tol = 1e-06, verbose, max.iter = 100, near.zero.var = FALSE})
\]

**Arguments**

- **X**
  - A list of data sets (called 'blocks’) measured on the same samples. Data in the list should be arranged in matrices, samples x variables, with samples order matching in all data sets.

- **Y**
  - Matrix response for a multivariate regression framework. Data should be continuous variables (see block.splsda for supervised classification and factor response)

- **indy**
  - To supply if Y is missing, indicates the position of the matrix response in the list X
ncomp
defines the number of components to include in the model. Default to 2. Applies to all blocks.

design
A numeric matrix of size (number of blocks in X) x (number of blocks in X) with values between 0 and 1. Each value indicates the strength of the relationship to be modelled between two blocks; a value of 0 indicates no relationship, 1 is the maximum value. If Y is provided instead of indY, the design matrix is changed to include relationships to Y.
scheme
Either "horst", "factorial" or "centroid". Default = horst, see reference.
mode
A character string. What type of algorithm to use, (partially) matching one of "regression", "canonical", "invariant" or "classic". See Details. Default = regression.
scale
A boolean. If scale = TRUE, each block is standardized to zero means and unit variances. Default = TRUE.
bias
A boolean. A logical value for biased or unbiased estimator of the var/cov. Default = FALSE.
init
Mode of initialization use in the algorithm, either by Singular Value Decomposition of the product of each block of X with Y ("svd") or each block independently ("svd.single"). Default = svd.single.
tol
Convergence stopping value.
verbose
if set to TRUE, reports progress on computing.
max.iter
integer, the maximum number of iterations.
near.zero.var
boolean, see the internal nearZeroVar function (should be set to TRUE in particular for data with many zero values). Default = FALSE.

Details

The block pls function fits a horizontal integration PLS model with a specified number of components per block. An outcome needs to be provided, either by Y or by its position indY in the list of blocks X. Multi (continuous)response are supported. X and Y can contain missing values. Missing values are handled by being disregarded during the cross product computations in the algorithm block.pls without having to delete rows with missing data. Alternatively, missing data can be imputed prior using the nipals function.

The type of algorithm to use is specified with the mode argument. Four PLS algorithms are available: PLS regression ("regression"), PLS canonical analysis ("canonical"), redundancy analysis ("invariant") and the classical PLS algorithm ("classic") (see References and ?pls for more details).

Note that our method is partly based on Generalised Canonical Correlation Analysis and differs from the MB-PLS approaches proposed by Kowalski et al., 1989, J Chemom 3(1) and Westerhuis et al., 1998, J Chemom, 12(5).

Value

block.pls returns an object of class "block.pls", a list that contains the following components:

X
the centered and standardized original predictor matrix.
block.pls

indY the position of the outcome Y in the output list X.
ncomp the number of components included in the model for each block.
mode the algorithm used to fit the model.
variates list containing the variates of each block of X.
loadings list containing the estimated loadings for the variates.
names list containing the names to be used for individuals and variables.
nzv list containing the zero- or near-zero predictors information.
iter Number of iterations of the algorithm for each component
explained_variance Percentage of explained variance for each component and each block

Author(s)
Florian Rohart, Benoit Gautier, Kim-Anh Le Cao

References
Tenenhaus A. and Tenenhaus M., (2011), Regularized Generalized Canonical Correlation Analysis,

See Also
plotIndiv, plotArrow, plotLoadings, plotVar, predict.perf, selectVar, block.spls, block.plsda
and http://www.mixOmics.org for more details.

Examples

# Example with TCGA multi omics study
# ----------------------------------
data("breast.TCGA")
# this is the X data as a list of mRNA and miRNA; the Y data set is a single data set of proteins
data = list(mrna = breast.TCGA$data.train$mrna, mirna = breast.TCGA$data.train$mirna)
# set up a full design where every block is connected
design = matrix(1, ncol = length(data), nrow = length(data),
dimnames = list(names(data), names(data)))
diag(design) = 0
design
# set number of component per data set
ncomp = c(2)

TCGA.block.pls = block.pls(X = data, Y = breast.TCGA$data.train$protein, ncomp = ncomp,
design = design)
TCGA.block.pls
# in plotindiv we color the samples per breast subtype group but the method is unsupervised!
# here Y is the protein data set
plotIndiv(TCGA.block.pls, group = breast.TCGA$data.train$subtype, ind.names = FALSE)

---

**block.plsda**

*N-integration with Projection to Latent Structures models (PLS) with Discriminant Analysis*

**Description**

Integration of multiple data sets measured on the same samples or observations to classify a discrete outcome, i.e. N-integration with Discriminant Analysis. The method is partly based on Generalised Canonical Correlation Analysis.

**Usage**

```
block.plsda(X, 
Y, 
indy, 
ncomp = 2, 
design, 
scheme, 
mode, 
scale = TRUE, 
bias, 
init, 
tol = 1e-06, 
verbose, 
max.iter = 100, 
near.zero.var = FALSE)
```

**Arguments**

- **X**: A list of data sets (called 'blocks') measured on the same samples. Data in the list should be arranged in matrices, samples x variables, with samples order matching in all data sets.

- **Y**: A factor or a class vector indicating the discrete outcome of each sample.

- **indy**: To be supplied if Y is missing, indicates the position of the factor / class vector outcome in the list X.

- **ncomp**: the number of components to include in the model. Default to 2. Applies to all blocks.

- **design**: numeric matrix of size (number of blocks in X) x (number of blocks in X) with values between 0 and 1. Each value indicates the strength of the relationship to be modelled between two blocks; a value of 0 indicates no relationship, 1 is the maximum value. If Y is provided instead of indY, the design matrix is changed to include relationships to Y.
block.plsda

scheme character string. Either "horst", "factorial" or "centroid". Default = horst, see reference.
mode character string. What type of algorithm to use. Partially matching one of "regression", "canonical", "invariant" or "classic". Default = regression.
scale boolean. If scale = TRUE, each block is standardized to zero means and unit variances. Default = TRUE.
bias boolean. A logical value for biaised or unbiaised estimator of the var/cov. Default = FALSE.
init Mode of initialization use in the algorithm, either Singular Value Decomposition of the product of each block of X with Y ("svd") or each block independently ("svd.single"). Default = svd.single.
tol Convergence stopping value.
verbose if set to TRUE, reports progress on computing.
max.iter integer, the maximum number of iterations.
near.zero.var boolean, see the internal nearZeroVar function (should be set to TRUE in particular for data with many zero values). Default = FALSE.

Details

block.plsda function fits a horizontal integration PLS-DA model with a specified number of components per block. A factor indicating the discrete outcome needs to be provided, either by Y or by its position indY in the list of blocks X.

X can contain missing values. Missing values are handled by being disregarded during the cross product computations in the algorithm block.pls without having to delete rows with missing data. Alternatively, missing data can be imputed prior using the nipals function.

The type of algorithm to use is specified with the mode argument. Four PLS algorithms are available: PLS regression ("regression"), PLS canonical analysis ("canonical"), redundancy analysis ("invariant") and the classical PLS algorithm ("classic") (see References and _pls for more details).

Note that our method is partly based on Generalised Canonical Correlation Analysis and differs from the MB-PLS approaches proposed by Kowalski et al., 1989, J Chemom 3(1) and Westerhuis et al., 1998, J Chemom, 12(5).

Value

block.plsda returns an object of class "block.plsda", "block.pls", a list that contains the following components:

X the centered and standardized original predictor matrix.
indY the position of the outcome Y in the output list X.
ncomp the number of components included in the model for each block.
mode the algorithm used to fit the model.
variates list containing the variates of each block of X.
loadings list containing the estimated loadings for the variates.
names  list containing the names to be used for individuals and variables.
rvz  list containing the zero- or near-zero predictors information.
iter  Number of iterations of the algorithm for each component
explained_variance  Percentage of explained variance for each component and each block

Author(s)
Florian Rohart, Benoit Gautier, Kim-Anh Le Cao

References
On PLS-DA:


On multiple integration with sPLS-DA and 4 data blocks:

mixOomics manuscript:

See Also
plotIndiv, plotArrow, plotLoadings, plotVar, predict, perf, selectVar, block.pls, block.plsda and http://www.mixOmics.org for more details.

Examples

data(nutrimouse)
data = list(gene = nutrimouse$gene, lipid = nutrimouse$lipid, Y = nutrimouse$diet)
# with this design, all blocks are connected
design = matrix(c(0,1,1,0,1,1,1,0), ncol = 3, nrow = 3,
byrow = TRUE, dimnames = list(names(data), names(data))
res = block.plsda(X = data, indY = 3) # indY indicates where the outcome Y is in the list X
plotIndiv(res, ind.names = FALSE, legend = TRUE)
**Description**

Integration of multiple data sets measured on the same samples or observations, with variable selection in each data set, i.e. N-integration. The method is partly based on Generalised Canonical Correlation Analysis.

**Usage**

```r
block.spls(X, 
Y, 
indY, 
ncomp = 2, 
keepX.constraint, 
keepY.constraint, 
keepX, 
keepY, 
design, 
scheme, 
mode, 
scale = TRUE, 
bias, 
init, 
tol = 1e-06, 
verbose, 
max.iter = 100, 
near.zero.var = FALSE)
```

**Arguments**

- **X**: A list of data sets (called 'blocks') measured on the same samples. Data in the list should be arranged in matrices, samples x variables, with samples order matching in all data sets.
Y

Matrix response for a multivariate regression framework. Data should be continuous variables (see block.splsda for supervised classification and factor response).

indY

To supply if Y is missing, indicates the position of the matrix response in the list X.

ncomp

the number of components to include in the model. Default to 2. Applies to all blocks.

keepX.constraint

A list of same length as X. Each entry indicates specific variables to be selected in each of the blocks of X for the first PLS-components. See details and example.

keepy.constraint

Only if Y is provided, otherwise extracted from keepX.constraint (see details). Each entry indicates specific variables to be selected in Y for the first PLS-components. See example.

keepX

A list of same length as X. Each entry is the number of variables to select in each of the blocks of X for each component. In the case where keepX.constraint is specified it should indicate the number of variables to select on the remaining components. By default all variables are kept in the model.

keepY

Only if Y is provided. Each entry is the number of variables to select in each of the blocks of Y for each component. In the case where keepX.constraint is specified it should indicate the number of variables to select on the remaining components.

design

numeric matrix of size (number of blocks in X) x (number of blocks in X) with values between 0 and 1. Each value indicates the strength of the relationship to be modelled between two blocks; a value of 0 indicates no relationship, 1 is the maximum value. If Y is provided instead of indY, the design matrix is changed to include relationships to Y.

scheme

Either "horst", "factorial" or "centroid". Default = horst, see reference.

mode

character string. What type of algorithm to use, (partially) matching one of "regression", "canonical", "invariant" or "classic". See Details. Default = regression.

scale

boolean. If scale = TRUE, each block is standardized to zero means and unit variances. Default = TRUE.

bias

boolean. A logical value for biased or unbiased estimator of the var/cov. Default = FALSE.

init

Mode of initialization use in the algorithm, either by Singular Value Decomposition of the product of each block of X with Y ("svd") or each block independently ("svd.single"). Default = svd.single.

tol

Convergence stopping value.

verbose

if set to TRUE, reports progress on computing.

max.iter

integer, the maximum number of iterations.

near.zero.var

boolean, see the internal nearZeroVar function (should be set to TRUE in particular for data with many zero values). Default = FALSE.
Details

The `block.spls` function fits a horizontal sPLS model with a specified number of components per block. An outcome needs to be provided, either by `Y` or by its position `indY` in the list of blocks `X`. Multi (continuous) response are supported. `X` and `Y` can contain missing values. Missing values are handled by being disregarded during the cross product computations in the algorithm `block.pls` without having to delete rows with missing data. Alternatively, missing data can be imputed prior using the `nipals` function.

The type of algorithm to use is specified with the `mode` argument. Four PLS algorithms are available: PLS regression ("regression"), PLS canonical analysis ("canonical"), redundancy analysis ("invariant") and the classical PLS algorithm ("classic") (see References and `?pls` for more details).

Note that our method is partly based on sparse Generalised Canonical Correlation Analysis and differs from the MB-PLS approaches proposed by Kowalski et al., 1989, J Chemom 3(1), Westerhuis et al., 1998, J Chemom, 12(5) and sparse variants Li et al., 2012, Bioinformatics 28(19); Karaman et al (2014), Metabolomics, 11(2); Kawaguchi et al., 2017, Biostatistics.

Constraint model. To improve performance, a new argument was recently added to fit a constraint model (constraint=TRUE). The model is run on the optimal list of selected features `keepX.constraint` from the previous components, as opposed to considering only a specified number of features `keepX`.

Variable selection is performed on each component for each block of `X`, and for `Y` if specified, via input parameter `keepX` and `keepY`, and `keepX.constraint` and `keepY.constraint` for a constraint model. `keepX.constraint` and `keepY.constraint` specify variable names to be selected in the previous components of the model (see example).

Note that if `Y` is missing and `indY` is provided, then variable selection on `Y` is performed by specifying the input parameter directly in `keepX.constraint` and `keepX` (no `keepY.constraint` and `keepY` is needed).

Value

`block.spls` returns an object of class "block.spls", a list that contains the following components:

- `X` the centered and standardized original predictor matrix.
- `indY` the position of the outcome `Y` in the output list `X`.
- `ncomp` the number of components included in the model for each block.
- `mode` the algorithm used to fit the model.
- `keepX` number of variables used to build each component of each block
- `keepY` number of variables used to build each component of `Y`
- `keepX.constraint` list indicating which variables (indices) where constrained to be used in the first components for each block in the `X` data set
- `keepY.constraint` list indicating which variables (indices) where constrained to be used in the first components for each block in the `Y` data set
- `variates` list containing the variates of each block of `X`. 
loadings  list containing the estimated loadings for the variates.
names  list containing the names to be used for individuals and variables.
nzv  list containing the zero- or near-zero predictors information.
iter  Number of iterations of the algorithm for each component
explained_variance  Percentage of explained variance for each component and each block

**Author(s)**
Florian Rohart, Benoit Gautier, Kim-Anh Le Cao

**References**
Tenenhaus A. and Tenenhaus M., (2011), Regularized Generalized Canonical Correlation Analysis,
regularized canonical correlation analysis. *Biostatistics*. kxu001

**See Also**
plotindiv, plotArrow, plotLoadings, plotVar, predict, perf, selectVar, block.pls, block.splsda
and http://www.mixOmics.org for more details.

**Examples**

```r
# First example with multi omics TCGA study
# ---------------------------------
data("breast.TCGA")
# this is the X data as a list of mRNA and miRNA; the Y data set is a single data set of proteins
data = list(mrna = breast.TCGA$data.train$mRNA, mirna = breast.TCGA$data.train$miRNA)
# set up a full design where every block is connected
design = matrix(1, ncol = length(data), nrow = length(data),
dimnames = list(names(data), names(data)))
diag(design) = 0
design
# set number of component per data set
ncomp = c(2)
# set number of variables to select, per component and per data set (this is set arbitrarily)
list.keepX = list(mrna = rep(20, 2), mirna = rep(10,2))
list.keepY = c(rep(10, 2))

TCGA.block.spls = block.spls(X = data, Y = breast.TCGA$data.train$protein,
ncomp = ncomp, keepX = list.keepX, keepY = list.keepY, design = design)
TCGA.block.spls
# in plotindiv we color the samples per breast subtype group but the method is unsupervised!
```
plotIndiv(TCGA.block.spls, group = breast.TCGA$data.train$subtype, ind.names = FALSE)
# illustrates coefficient weights in each block
plotLoadings(TCGA.block.spls, ncomp = 1)
plotVar(TCGA.block.spls, style = 'graphics', legend = TRUE)
network(TCGA.block.spls)

# Second example with a constraint sPLS
# ----------------------
# using the data and design from example above
## we force the first PLS-component of X to be built with only a few variables selected
# in the earlier example

constraint.mrna.compl = c("EPHB3", "CBR1", "BSPRY")
constraint.mirna.compl = c("hsa-mir-128-2", "hsa-mir-196b")
constraint.prot.compl = c("Paxillin")

TCGA.block.spls.constraint = block.spls(X = data, Y = breast.TCGA$data.train$protein, ncomp = 2,
keepX = list(mrna = 20, mirna = 20),
keepY = 20,
keepX.constraint = list(mrna = list(constraint.mrna.compl),
mirna = list(constraint.mirna.compl)),
keepY.constraint = list(protein = list(constraint.prot.compl)))

# see if that subset contraint on comp 1 affected the variables selected in comp 2:
intersect(selectVar(TCGA.block.spls.constraint, comp = 2)$mrna$name,
selectVar(TCGA.block.spls, comp = 2)$mrna$name)

# indices of the constraint variable on the first component
TCGA.block.spls.constraint$keepX.constraint
TCGA.block.spls.constraint$keepY.constraint

---

**block.splSDA**

N-integration and feature selection with Projection to Latent Structures models (PLS) with sparse Discriminant Analysis

**Description**

Integration of multiple data sets measured on the same samples or observations to classify a discrete outcome to classify a discrete outcome and select features from each data set, ie. N-integration with sparse Discriminant Analysis. The method is partly based on Generalised Canonical Correlation Analysis.

**Usage**

```r
block.splSDA(X,
Y,
indY,
ncomp = 2,
keepX.constraint,
```
keepX,
design,
scheme,
mode,
scale = TRUE,
bias,
init,
tol = 1e-06,
verbose,
max.iter = 100,
near.zero.var = FALSE)

Arguments

X A list of data sets (called 'blocks') measured on the same samples. Data in the list should be arranged in matrices, samples x variables, with samples order matching in all data sets.

Y A factor or a class vector indicating the discrete outcome of each sample.

indy To be supplied if Y is missing, indicates the position of the factor / class vector outcome in the list X

ncomp the number of components to include in the model. Default to 2. Applies to all blocks.

keepX.constraint A list of same length as X. Each entry indicates specific variables to be selected in each of the blocks of X for the first PLS-components. See details and example.

keepX A list of same length as X. Each entry is the number of variables to select in each of the blocks of X for each component. In the case where keepX.constraint is specified it should indicate the number of variables to select on the remaining components. By default all variables are kept in the model.

design numeric matrix of size (number of blocks in X) x (number of blocks in X) with values between 0 and 1. Each value indicates the strenght of the relationship to be modelled between two blocks; a value of 0 indicates no relationship, 1 is the maximum value. If Y is provided instead of indY, the design matrix is changed to include relationships to Y.

scheme Either "horst", "factorial" or "centroid". Default = horst, see reference.

mode character string. What type of algorithm to use, (partially) matching one of "regression", "canonical", "invariant" or "classic". See Details. Default = regression.

scale booleean. If scale = TRUE, each block is standardized to zero means and unit variances. Default = TRUE.

bias booleean. A logical value for biaised or unbiaised estimator of the var/cov. Default = FALSE.

init Mode of initialization use in the algorithm, either by Singular Value Decomposition of the product of each block of X with Y ("svd") or each block independently ("svd.single"). Default = svd.single.
tol  Convergence stopping value.
verbose if set to TRUE, reports progress on computing.
max.iter integer, the maximum number of iterations.
near.zero.var boolean, see the internal nearZeroVar function (should be set to TRUE in particular for data with many zero values). Default = FALSE.

Details

block.splsda function fits a horizontal integration PLS-DA model with a specified number of components per block). A factor indicating the discrete outcome needs to be provided, either by Y or by its position indY in the list of blocks X.

X can contain missing values. Missing values are handled by being disregarded during the cross product computations in the algorithm block.pls without having to delete rows with missing data. Alternatively, missing data can be imputed prior using the nipals function.

The type of algorithm to use is specified with the mode argument. Four PLS algorithms are available: PLS regression ("regression"), PLS canonical analysis ("canonical"), redundancy analysis ("invariant") and the classical PLS algorithm ("classic") (see References and _pls for more details).

Note that our method is partly based on sparse Generalised Canonical Correlation Analysis and differs from the MB-PLS approaches proposed by Kowalski et al., 1989, J Chemom 3(1), Westerhuis et al., 1998, J Chemom, 12(5) and sparse variants Li et al., 2012, Bioinformatics 28(19); Karaman et al (2014), Metabolomics, 11(2); Kawaguchi et al., 2017, Biostatistics.

Constraint model. To improve performance, a new argument was recently added to fit a constraint model (constraint=TRUE). The model is run on the optimal list of selected features keepX.constraint from the previous components, as opposed to considering only a specified number of features keepX. Our experience has shown that the constraint tuning and models can substantially improve the performance of the methods (see also tune.block.splsda)

Variable selection is performed on each component for each block of X if specified, via input parameter keepX and keepX.constraint for a constraint model. keepX.constraint specifies variable names to be selected in the previous components of the model (see example).

Value

block.splsda returns an object of class "block.splsda", "block.pls", a list that contains the following components:

X the centered and standardized original predictor matrix.
indY the position of the outcome Y in the output list X.
ncomp the number of components included in the model for each block.
mode the algorithm used to fit the model.
keepX Number of variables used to build each component of each block
keepX.constraint list indicating which variables (indices) where constrained to be used to build the first component for each block

variates list containing the variates of each block of X.
loadings list containing the estimated loadings for the variates.
names list containing the names to be used for individuals and variables.
nzv list containing the zero- or near-zero predictors information.
iter Number of iterations of the algorithm for each component
weights Correlation between the variate of each block and the variate of the outcome. Used to weight predictions.
explained_variance Percentage of explained variance for each component and each block

Author(s)
Florian Rohart, Benoit Gautier, Kim-Anh Le Cao

References
On multiple integration with sPLS-DA and 4 data blocks:

On data integration:
mixOmics manuscript:

See Also
plotIndiv, plotArrow, plotLoadings, plotVar, predict, perf, selectVar, blockNplsda, block.spls
and http://www.mixOmics.org/mixDIABLO for more details and examples.

Examples
# block.splsda
# ------------
data("breast.TCGA")
# this is the X data as a list of mRNA, miRNA and proteins
data = list(mrna = breast.TCGA$data.train$mRNA, mirna = breast.TCGA$data.train$miRNA, protein = breast.TCGA$data.train$protein)
# set up a full design where every block is connected
design = matrix(1, ncol = length(data), nrow = length(data),
dimnames = list(names(data), names(data)))


```r
diag(design) = 0
design
# set number of component per data set
ncomp = c(2)
# set number of variables to select, per component and per data set (this is set arbitrarily)
list.keepX = list(mrna = rep(20, 2), mirna = rep(10, 2), protein = rep(10, 2))

TCGA.block.splsda = block.splsda(X = data, Y = breast.TCGA$data.train$subtype,
ncomp = ncomp, keepX = list.keepX, design = design)

plotIndiv(TCGA.block.splsda, ind.names = FALSE)
# illustrates coefficient weights in each block
plotLoadings(TCGA.block.splsda, ncomp = 1, contrib = 'max')
plotVar(TCGA.block.splsda, style = 'graphics', legend = TRUE)

# block.splsda constraint
# ---------------
# using the data and design from example above
## we force the first PLS-component of X to be built with only a few variables
# selected in the earlier example

constraint.mrna.comp1 = c("EPHB3", "CBR1", "BSPRY")
constraint.mirna.comp1 = c("hsa-mir-128-2", "hsa-mir-196b")
constraint.prot.compl = c("Paxillin")

TCGA.block.splsda.constraint = block.splsda(X = data, Y = breast.TCGA$data.train$subtype, ncomp = 2,
keepX = list(mrna = 20, mirna = 20, protein = 10),
keepX.constraint = list(mrna = list(constraint.mrna.comp1), mirna = list(constraint.mirna.comp1),
protein = list(constraint.prot.compl)))

# see if that subset contraint on comp 1 affected the variables selected in comp 2:
intersect(selectVar(TCGA.block.splsda.constraint, comp = 2)$mrna$name,
selectVar(TCGA.block.splsda, comp = 2)$mrna$name)

# indices of the constraint variables on the first component
TCGA.block.splsda.constraint$keepX.constraint
```

---

**breast.TCGA**

*Breast Cancer multi omics data from TCGA*

---

**Description**

This data set is a small subset of the full data set from The Cancer Genome Atlas that can be analysed with the DIABLO framework. It contains the expression or abundance of three matching omics data sets: mRNA, miRNA and proteomics for 150 breast cancer samples (Basal, Her2, Luminal A) in the training set, and 70 samples in the test set. The test set is missing the proteomics data set.
Usage

data(breast.TCGA)

Format

A list containing two data sets, data.train and data.test which both include:

miRNA data frame with 150 (70) rows and 184 columns in the training (test) data set. The expression levels of 184 miRNA.

mRNA data frame with 150 (70) rows and 520 columns in the training (test) data set. The expression levels of 200 mRNA.

protein data frame with 150 (70) rows and 142 columns in the training data set only. The abundance of 142 proteins.

subtype a factor indicating the breast cancer subtypes in the training (length of 150) and test (length of 70) sets.

Details

The data come from The Cancer Genome Atlas (TCGA, http://cancergenome.nih.gov/). We divided the data into a training (discovery) and test (validation) set. The protein dataset which had a limited number of subjects available was used to allocate subjects into the training set only, while the test set included all remaining subject. Each data set was normalised and pre-processed. For illustrative purposes we drastically filtered the data here.

Source

The raw data were downloaded from http://cancergenome.nih.gov/. The normalised and filtered data we analysed with DIABLO are available on www.mixOmics.org/mixDIABLO

References

**Format**

A list containing the following components:

- `gene.exp` data matrix with 47 rows and 1000 columns. Each row represents an experimental sample, and each column a single gene.
- `sample` a list containing two character vector components: `name` the name of the samples, and `treatment` the treatment status.
- `genes` a list containing two character vector components: `name` the name of the genes, and `description` the description of each gene.

**Details**

This data consists of 47 breast cancer samples and 1753 cDNA clones pre-selected by Perez-Enciso et al. (2003) to draw their Fig. 1. The authors selected 47 samples for which there was information at least before or before and after chemotherapy treatment. There were 20 tumours that were microarrayed both before and after treatment. For illustrative purposes we then randomly selected 1000 cDNA clones for this data set.

**Source**

The Human Breast Tumors dataset is a companion resource for the paper of Perou et al. (2000), and was downloaded from the Stanford Genomics Breast Cancer Consortium Portal [http://genome-www.stanford.edu/breast_cancer/molecularportraits/download.shtml](http://genome-www.stanford.edu/breast_cancer/molecularportraits/download.shtml)

**References**


---

**cim**

*Clustered Image Maps (CIMs) ("heat maps")*

**Description**

This function generates color-coded Clustered Image Maps (CIMs) ("heat maps") to represent "high-dimensional" data sets.
Usage

cim(mat,    
color = NULL,    
row.names = TRUE,    
col.names = TRUE,    
row.sideColors = NULL,    
col.sideColors = NULL,    
row.cex = NULL,    
col.cex = NULL,    
threshold = 0,    
cluster = "both",    
dist.method = c("euclidean", "euclidean"),    
clust.method = c("complete", "complete"),    
cut.tree = c(0, 0),    
transpose = FALSE,    
symkey = TRUE,    
keysize = c(1, 1),    
keysize.label = 1,    
zoom = FALSE,    
title = NULL,    
xlab = NULL,    
ylab = NULL,    
margins = c(5, 5),    
lhei = NULL,    
lwid = NULL,    
comp= NULL,    
center = TRUE,    
scale = FALSE,    
mapping = "xy",    
legend= NULL,    
save = NULL,    
name.save = NULL)

Arguments

mat numeric matrix of values to be plotted. Alternatively, an object of class inher-    iting from "pca", "spca", "ipca", "sipca", "rcc", "pls", "spls", "plsd"a",    "splsda", "mlspls" or "mlsplsd".
color a character vector of colors such as that generated by terrain.colors, topo.colors,    rainbow, color.jet or similar functions.
row.names, col.names logical, should the name of rows and/or columns of mat be shown? If TRUE    (defaults) rownames(mat) and/or colnames(mat) are used. Possible character vectors with row and/or column labels can be used.
row.sideColors (optional) character vector of length nrow(mat) containing the color names for    a vertical side bar that may be used to annotate the rows of mat.
col.sideColors (optional) character vector of length ncol(mat) containing the color names for    a horizontal side bar that may be used to annotate the columns of mat.
positive numbers, used as cex.axis in for the row or column axis labeling. The
defaults currently only use number of rows or columns, respectively.

mapping character string indicating whether to map "X", "Y" or "XY"-association matrix.
See Details.

cluster character string indicating whether to cluster "none", "row", "column" or "both".
Defaults to "both".

dist.method character vector of length two. The distance measure used in clustering rows
and columns. Possible values are "correlation" for Pearson correlation and
all the distances supported by dist, such as "euclidean", etc.

clust.method character vector of length two. The agglomeration method to be used for rows
and columns. Accepts the same values as in hclust such as "ward", "complete", etc.

cut.tree numeric vector of length two with components in [0,1]. The height proportions
where the trees should be cut for rows and columns, if these are clustered.

comp atomic or vector of positive integers. The components to adequately account
for the data association. For a non sparse method, the similarity matrix is
computed based on the variates and loading vectors of those specified compo-
nents. For a sparse approach, the similarity matrix is computed based on the
variables selected on those specified components. See example. Defaults to
comp = 1:object$ncomp.

transpose logical indicating if the matrix should be transposed for plotting. Defaults to
FALSE.

center either a logical value or a numeric vector of length equal to the number of
columns of mat. See scale function.

scale either a logical value or a numeric vector of length equal to the number of
columns of mat. See scale function.

threshold numeric between 0 and 1. Variables with correlations below this threshold in
absolute value are not plotted. To use only when mapping is "XY".

symkey boolean indicating whether the color key should be made symmetric about 0.
Defaults to TRUE.

keysize vector of length two, indicating the size of the color key.

keysize.label vector of length 1, indicating the size of the labels and title of the color key.

zoom logical. Whether to use zoom for interactive zoom. See Details.

title, xlab, ylab title, x- and y-axis titles; default to none.

margins numeric vector of length two containing the margins (see par(par)) for column
and row names respectively.

lhei, lwid arguments passed to layout to divide the device up into two (or three if a
side color is drawn) rows and two columns, with the row-heights lhei and the
column-widths lwid.

legend A list indicating the legend for each group, the color vector, title of the legend
and cex.
save should the plot be saved? If so, argument to be set to either 'jpeg', 'tiff', 'png' or 'pdf'.

name.save character string for the name of the file to be saved.

Details

One matrix Clustered Image Map (default method) is a 2-dimensional visualization of a real-valued matrix (basically image(t(mat))) with rows and/or columns reordered according to some hierarchical clustering method to identify interesting patterns. Generated dendrograms from clustering are added to the left side and to the top of the image. By default the used clustering method for rows and columns is the complete linkage method and the used distance measure is the distance euclidean.

In "pca", "spca", "ipca", "sipca", "plsda", "splsda" and "mlsplsda" methods the mat matrix is object$X.

For the remaining methods, if mapping = "X" or mapping = "Y" the mat matrix is object$X or object$Y respectively. If mapping = "XY":

- in rcc method, the matrix mat is created where element (j, k) is the scalar product value between every pairs of vectors in dimension length(comp) representing the variables $X_j and $Y_k on the axis defined by $Z_i with i in comp, where $Z_i is the equiangular vector between the i-th $X and $Y canonical variate.

- in pls, spls and mlspls methods, if object$mode is "regression", the element (j, k) of the matrix mat is given by the scalar product value between every pairs of vectors in dimension length(comp) representing the variables $X_j and $Y_k on the axis defined by $U_i with i in comp, where $U_i is the i-th $X variate. If object$mode is "canonical" then $X_j and $Y_k are represented on the axis defined by $U_i and $V_i respectively.

By default four components will be displayed in the plot. At the top left is the color key, top right is the column dendogram, bottom left is the row dendogram, bottom right is the image plot. When sideColors are provided, an additional row or column is inserted in the appropriate location. This layout can be overriden by specifying appropriate values for lwid and lhei. lwid controls the column width, and lhei controls the row height. See the help page for layout for details on how to use these arguments.

For visualization of "high-dimensional" data sets, a nice zooming tool was created. zoom = TRUE open a new device, one for CIM, one for zoom-out region and define an interactive 'zoom' process: click two points at imagen map region by pressing the first mouse button. It then draws a rectangle around the selected region and zoom-out this at new device. The process can be repeated to zoom-out other regions of interest.

The zoom process is terminated by clicking the second button and selecting 'Stop' from the menu, or from the 'Stop' menu on the graphics window.

Value

A list containing the following components:

M the mapped matrix used by cim.

rowInd, colInd row and column index permutation vectors as returned by order.dendrogram.
**ddr, ddc** object of class "dendrogram" which describes the row and column trees produced by cim.

**mat.cor** the correlation matrix used for the heatmap. Available only when mapping = "XY".

**row.names, col.names** character vectors with row and column labels used.

**row.sideColors, col.sideColors** character vector containing the color names for vertical and horizontal side bars used to annotate the rows and columns.

**Author(s)**

Ignacio Gonzalez, Francois Bartolo, Kim-Anh Le Cao.

**References**


**See Also**

heatmap, hclust, plotVar, network and http://mixomics.org/graphics/ for more details on all options available.

**Examples**

```r
# default method: shows cross correlation between 2 data sets
#---------------------------------------------------------------
data(nutrimouse)
X <- nutrimouse$lipid
Y <- nutrimouse$gene
cim(cor(X, Y), cluster = "none")

# CIM representation for objects of class 'rcc'
#---------------------------------------------------------------
## Not run:
```

```r
```
nutri.rcc <- rcc(X, Y, ncomp = 3, lambda1 = 0.064, lambda2 = 0.008)
cim(nutri.rcc, xlab = "genes", ylab = "lipids", margins = c(5, 6))

## End(Not run)

## interactive 'zoom' available as below
## Not run:

cim(nutri.rcc, xlab = "genes", ylab = "lipids", margins = c(5, 6),
    zoom = TRUE)

##-- select the region and "see" the zoom-out region

##-- cim from X matrix with a side bar to indicate the diet
diet.col <- palette()[as.numeric(nutrimouse$diet)]
cim(nutri.rcc, mapping = "X", row.names = nutrimouse$diet,
    row.sideColors = diet.col, xlab = "lipids",
    clust.method = c("ward", "ward"), margins = c(6, 4))

##-- cim from Y matrix with a side bar to indicate the genotype
geno.col = color.mixo(as.numeric(nutrimouse$genotype))
cim(nutri.rcc, mapping = "Y", row.names = nutrimouse$genotype,
    row.sideColors = geno.col, xlab = "genes",
    clust.method = c("ward", "ward"))

##-- save the result as a jpeg file
jpeg(filename = "test.jpeg", res = 600, width = 4000, height = 4000)
cim(nutri.rcc, xlab = "genes", ylab = "lipids", margins = c(5, 6))
dev.off()

## End(Not run)

## CIM representation for objects of class 'spca' (also works for sipca)
#---------------------------------------------------------------
## Not run:
data(liver.toxicity)
X <- liver.toxicity$gene

liver.spca <- spca(X, ncomp = 2, keepX = c(30, 30), scale = FALSE)
dose.col <- color.mixo(as.numeric(as.factor(liver.toxicity$treatment[, 3])))

# side bar, no variable names shown
cim(liver.spca, row.sideColors = dose.col, col.names = FALSE,
    row.names = liver.toxicity$treatment[, 3],
    clust.method = c("ward", "ward"))

## End(Not run)

## CIM representation for objects of class '(s)pls'
#---------------------------------------------------------------
## Not run:
data(liver.toxicity)
```r
X <- liver.toxicity$gene
Y <- liver.toxicity$clinic
liver.spls <- spls(X, Y, ncomp = 3,
        keepX = c(20, 50, 50), keepY = c(10, 10, 10))

# default
cim(liver.spls)

# transpose matrix, choose clustering method
cim(liver.spls, transpose = TRUE,
    clust.method = c("ward", "ward"), margins = c(5, 7))

# Here we visualise only the X variables selected
cim(liver.spls, mapping="X")

# Here we should visualise only the Y variables selected
cim(liver.spls, mapping="Y")

# Here we only visualise the similarity matrix between the variables by spls
cim(liver.spls, cluster="none")

# plotting two data sets with the similarity matrix as input in the function
# (see our BioData Mining paper for more details)
# Only the variables selected by the sPLS model in X and Y are represented
# on the X matrix only, side col var to indicate dose
# with a sparse method, show only the variables selected on specific components
dose.col <- color.mixo(as.numeric(as.factor(liver.toxicity$treatment[, 3])))
cim(liver.spls, mapping = "X", row.sideColors = dose.col,
    row.names = liver.toxicity$treatment[, 3])

# CIM default representation includes the total of 120 genes selected, with the dose color
# with a sparse method, show only the variables selected on specific components
cim(liver.spls, comp = 1)
cim(liver.spls, comp = 2)
cim(liver.spls, comp = c(1,2))
cim(liver.spls, comp = c(1,3))

## End(Not run)
```
dose.col <- color.mixo(as.numeric(as.factor(liver.toxicity$treatment[, 3])))

liver.splsda <- splsda(X, Y, ncomp = 2, keepX = c(40, 30))

cim(liver.splsda, row.sideColors = dose.col, row.names = Y)

## End(Not run)

```
## CIM representation for objects of class splsda 'multilevel'
# with a two level factor (repeated sample and time)
#-----------------------------------------------
data(vac18.simulated)
X <- vac18.simulated$genes
design <- data.frame(samp = vac18.simulated$sample)
Y = data.frame(time = vac18.simulated$time,
               stim = vac18.simulated$stimulation)
res.2level <- splsda(X, Y, ncomp = 2, multilevel = design,
                     keepX = c(120, 10))

# define colors for the levels: stimulation and time
stim.col <- c("darkblue", "purple", "green4","red3")
time.col <- stim.col[as.numeric(Y$stim)]

# The row side bar indicates the two levels of the factor, stimulation and time.
# The sample names have been modified on the plot.
cim(res.2level, row.sideColors = cbind(stim.col, time.col),
     row.names = paste(Y$time, Y$stim, sep = "_")
     col.names = FALSE,
     legend=list(legend = c(levels(Y$time), levels(Y$stim)),
                      col = c("orange", "cyan", "darkblue", "purple", "green4","red3"),
                      title = "Condition", cex = 0.7))
```

## CIM representation for objects of class spls 'multilevel'
#---------------------------------------------------------
data(liver.toxicity)
repeat.indiv <- c(1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 3, 3, 4, 4, 3, 4, 3, 4, 5, 6, 5, 5,
                  6, 5, 6, 7, 7, 8, 6, 7, 8, 7, 8, 9, 10, 9, 10, 11, 9, 11,
                  10, 11, 12, 12, 10, 11, 12, 11, 12, 11, 12, 13, 14, 13, 14,
                  13, 14, 15, 16, 15, 16, 15, 16, 15, 16, 15)

# sPLS is a non supervised technique, and so we only indicate the sample repetitions
# in the design (1 factor only here, sample)
# sPLS takes as an input 2 data sets, and the variables selected
design <- data.frame(sample = repeat.indiv)
Clustered Image Maps (CIMs) ("heat maps") for DIABLO

Description

This function generates color-coded Clustered Image Maps (CIMs) ("heat maps") to represent "high-dimensional" data sets analysed with DIABLO.

Usage

```r
CIM_Diablo(object, 
    color = NULL, 
    color.Y, 
    color.blocks, 
    comp = NULL, 
    margins = c(2, 15), 
    legend.position = "topright", 
    transpose = FALSE,
)```

```r
res.spls.1level <- spls(X = liver.toxicity$gene, 
    Y=liver.toxicity$clinic, 
    multilevel = design, 
    ncomp = 2, 
    keepX = c(50, 50), keepY = c(5, 5), 
    mode = 'canonical')
stim.col <- c("darkblue", "purple", "green4", "red3")

# showing only the Y variables, and only those selected in comp 1
CIM(res.spls.1level, mapping="Y", 
    row.sideColors = stim.col[factor(liver.toxicity$treatment[,3])], comp = 1, 
    #setting up legend: 
    legend=list(legend = unique(liver.toxicity$treatment[,3]), col=stim.col, 
    title = "Dose", cex=0.9))

## Not run:
# showing only the X variables, for all selected on comp 1 and 2
CIM(res.spls.1level, mapping="X", 
    row.sideColors = stim.col[factor(liver.toxicity$treatment[,3])], 
    #setting up legend: 
    legend=list(legend = unique(liver.toxicity$treatment[,3]), col=stim.col, 
    title = "Dose", cex=0.9))

# These are the cross correlations between the variables selected in X and Y. 
# The similarity matrix is obtained as in our paper in Data Mining
CIM(res.spls.1level, mapping="XY")

## End(Not run)
row.names = TRUE,
col.names = TRUE,
size.legend = 1.5)

Arguments

object An object of class inheriting from "block.splsda".
color a character vector of colors such as that generated by terrain.colors, topo.colors,
rainbow, color.jet or similar functions.
color.Y a character vector of colors to be used for the levels of the outcome
color.blocks a character vector of colors to be used for the blocks
comp positive integer. The similarity matrix is computed based on the variables selected on those specified components. See example. Defaults to comp = 1.
margins numeric vector of length two containing the margins (see par(mar)) for column and row names respectively.
legend.position position of the legend, one of "bottomright", "bottom", "bottomleft", "left", "topleft", "top", "topright", "right" and "center".
transpose logical indicating if the matrix should be transposed for plotting. Defaults to FALSE.
row.names, col.names logical, should the name of rows and/or columns of mat be shown? If TRUE (defaults) rownames(mat) and/or colnames(mat) are used. Possible character vectors with row and/or column labels can be used.
size.legend size of the legend

Details

This function is a small wrapper of link{cim} specific to the DIABLO framework.

Author(s)

Amrit Singh, Florian Rohart

References


mixOms manuscript:

See Also

cim, heatmap, hclust, plotVar, network and
[http://mixomics.org/mixDIABLO/](http://mixomics.org/mixDIABLO/) for more details on all options available.

Examples

```r
## default method: shows cross correlation between 2 data sets
#-----------------------------------------------
data(nutrimouse)
Y = nutrimouse$diet
data = list(gene = nutrimouse$gene, lipid = nutrimouse$lipid)
design = matrix(c(0,1,1,0,1,1,0), ncol = 3, nrow = 3, byrow = TRUE)
nutrimouse.sgccda <- block.splsda(X = data, 
Y = Y,
design = design,
keepX = list(gene = c(10,10), lipid = c(15,15)),
ncomp = 2,
scheme = "centroid",
verbose = FALSE,
bias = FALSE)
cimDiablo(nutrimouse.sgccda)
```

---

circosPlot

circosPlot for DIABLO

Description

Displays variable correlation among different blocks

Usage

circosPlot(object,
comp = 1 : min(object$ncomp),
cutoff,
color.Y,
color.blocks,
color.cor,
var.names = NULL,
showIntraLinks = FALSE,
line = TRUE,
size.legend = 0.8,
ncol.legend = 1,
size.variables = 0.25,
size.labels = 1,
legend = TRUE)

Arguments

object An object of class inheriting from "block.splsda".
comp Numeric vector indicating which component to plot. Default to all
cutoff Only shows links with a correlation higher than cutoff
color.Y a character vector of colors to be used for the levels of the outcome
color.blocks a character vector of colors to be used for the blocks
color.cor a character vector of two colors. First one is for the negative correlation, second one is for the positive correlation
var.names Optional parameter. A list of length the number of blocks in object$X, containing the names of the variables of each block. If NULL, the colnames of the data matrix are used.
showIntraLinks if TRUE, shows the correlation higher than the threshold inside each block.
line if TRUE, shows the overall expression of the selected variables. see examples.
size.legend size of the legend
ncol.legend number of columns for the legend
size.variables size of the variable labels
size.labels size of the block labels
legend boolean. Whether the legend should be added. Default is TRUE.

Details
circosPlot function depicts correlations of variables selected with block.splsda among different blocks, using a generalisation of the method presented in Gonzalez et al 2012. If ncomp is specified, then only the variables selected on that component are displayed.

Value

If saved in an object, the circos plot will output the similarity matrix and the names of the variables displayed on the plot (see attributes(object)).

Author(s)

Michael Vacher, Amrit Singh, Florian Rohart, Kim-Anh Le Cao
References


mixOmics manuscript:


See Also

block.splsda, references and http://www.mixOmics.org/mixDIABLO for more details.

Examples

data(nutrimouse)
Y = nutrimouse$diet
data = list(gene = nutrimouse$gene, lipid = nutrimouse$lipid)
design = matrix(c(0,1,1,1,0,1,1,1,0), ncol = 3, nrow = 3, byrow = TRUE)
nutrimouse.sgccda <- wrapper.sgccda(X = data,
Y = Y, 
design = design,
keepX = list(gene = c(10,10), lipid = c(15,15)),
ncomp = 2,
scheme = "horst",
verbose = FALSE,
bias = FALSE)
circosPlot(nutrimouse.sgccda, cutoff = 0.7, ncol.legend = 2, size.legend = 1.1)
circosPlot(nutrimouse.sgccda, cutoff = 0.7, ncol.legend = 2, size.legend = 1.1,
color.Y = 1:5, color.blocks = c("green","brown"), color.cor = c("magenta", "purple"))
par(mfrow=c(2,2))
circosPlot(nutrimouse.sgccda, cutoff = 0.7, ncol.legend = 2,
  size.legend = 1.1)
circosPlot(nutrimouse.sgccda, cutoff = 0.7, ncol.legend = 2,
  size.legend = 1.1, showIntraLinks = TRUE)
circosPlot(nutrimouse.sgccda, cutoff = 0.7, ncol.legend = 1,
  size.legend = 1.1, showIntraLinks = TRUE)
circosPlot(nutrimouse.sgccda, cutoff = 0.7, ncol.legend = 2,
  size.legend = 1.1, showIntraLinks = TRUE, line = FALSE, size.variables = 0.5)
color.jet

Color Palette for mixOmics

Description

The functions create a vector of n "contiguous" colors (except the color.mixo which are colors used internally to fit our logo colors).

Usage

color.jet(n, alpha = 1)
color.spectral(n, alpha = 1)
color.GreenRed(n, alpha = 1)
color.mixo(num.vector)

Arguments

- **n**: an integer, the number of colors (≥ 1) to be in the palette.
- **alpha**: a numeric value between 0 and 1 for alpha channel (opacity).
- **num.vector**: for color.mixo an integer vector specifying which colors to use in the mixOmics palette (there are only 10 colors available).

Details

The function color.jet(n) create color scheme, beginning with dark blue, ranging through shades of blue, cyan, green, yellow and red, and ending with dark red. This colors palette is suitable for displaying ordered (symmetric) data, with n giving the number of colors desired.

Value

For color.jet(n), color.spectral(n), color.GreenRed(n) a character vector, cv, of color names. This can be used either to create a user-defined color palette for subsequent graphics by palette(cv), a col= specification in graphics functions or in par.

For color.mixo, a vector of colors matching the mixOmics logo (10 colors max.)

See Also

colorRamp, palette, colors for the vector of built-in "named" colors; hsv, gray, rainbow, terrain.colors, ... to construct colors; and heat.colors, topo.colors for images.

Examples

```r
# --------------
# jet colors
# --------------
par(mfrow = c(3, 1))
z <- seq(-1, 1, length = 125)
```
for (n in c(11, 33, 125)) {
    image(matrix(z, ncol = 1), col = color.jet(n),
          xaxt = 'n', yaxt = 'n', main = paste('n = ', n))
    box()
    par(usr = c(-1, 1, -1, 1))
    axis(1, at = c(-1, 0, 1))
}

# ------------------------
# spectral colors
# ------------------------
par(mfrow = c(3, 1))
z <- seq(-1, 1, length = 125)
for (n in c(11, 33, 125)) {
    image(matrix(z, ncol = 1), col = color.spectral(n),
          xaxt = 'n', yaxt = 'n', main = paste('n = ', n))
    box()
    par(usr = c(-1, 1, -1, 1))
    axis(1, at = c(-1, 0, 1))
}

# ------------------------
# GreenRed colors
# ------------------------
par(mfrow = c(3, 1))
z <- seq(-1, 1, length = 125)
for (n in c(11, 33, 125)) {
    image(matrix(z, ncol = 1), col = color.GreenRed(n),
          xaxt = 'n', yaxt = 'n', main = paste('n = ', n))
    box()
    par(usr = c(-1, 1, -1, 1))
    axis(1, at = c(-1, 0, 1))
}

# # ------------------------
# mixOmics colors
# # ------------------------
data(nutrimouse)
X <- nutrimouse$lipid
Y <- nutrimouse$gene
nutri.res <- rcc(X, Y, ncomp = 3, lambda1 = 0.064, lambda2 = 0.008)

my.colors = color.mixo(1:5)
my.pch = ifelse(nutrimouse$genotype == 'wt', 16, 17)
#plotIndiv(nutri.res, ind.names = FALSE, group = my.colors, pch = my.pch, cex = 1.5)
Description

The 16S data from the Human Microbiome Project includes only the most diverse body sites: Antecubital fossa (skin), Stool and Subgingival plaque (oral) and can be analysed using a multilevel approach to account for repeated measurements using our module mixMC. The data include 162 samples (54 unique healthy individuals) measured on 1,674 OTUs.

Usage

data(diverse.16S)

Format

A list containing two data sets, data.TSS and data.raw and some metadata information:

data.TSS data frame with 162 rows (samples) and 1674 columns (OTUs). The prefiltered normalised data using Total Sum Scaling normalisation.

data.raw data frame with 162 rows (samples) and 1674 columns (OTUs). The prefiltered raw count OTU data which include a 1 offset (i.e. no 0 values).

taxonomy data frame with 1674 rows (OTUs) and 6 columns indicating the taxonomy of each OTU.

Indiv data frame with 162 rows indicating sample meta data.

bodysite factor of length 162 indicating the bodysite with levels "Antecubital_fossa", "Stool" and "Subgingival_plaque".

sample vector of length 162 indicating the unique individual ID, useful for a multilevel approach to taken into account the repeated measured on each individual.

Details

The data were downloaded from the Human Microbiome Project (HMP, http://hmpdacc.org/HMQCP/all/ for the V1-3 variable region). The original data contained 43,146 OTU counts for 2,911 samples measured from 18 different body sites. We focused on the first visit of each healthy individual and focused on the three most diverse habitats. The prefiltered dataset included 1,674 OTU counts. We strongly recommend to use log ratio transformations on the data.TSS normalised data, as implemented in the PLS and PCA methods, see details on www.mixOmics.org/mixMC.

The data.raw include a 1 offset in order to be log ratios transformed after TSS normalisation. Consequently, the data.TSS are TSS normalisation of data.raw. The CSS normalisation was performed on the original data (including zero values)

Source

The raw data were downloaded from http://hmpdacc.org/HMQCP/all/. Filtering and normalisation described in our website www.mixOmics.org/mixMC

References

estim.regul  Estimate the parameters of regularization for Regularized CCA

Description
This function has been renamed tune.rcc, see tune.rcc.

explained_variance  Calculation of explained variance

Description
This function calculates the variance explained by variates.

Usage
explained_variance(data, variates, ncomp)

Arguments
- data: numeric matrix of predictors
- variates: variates as obtained from a pls object for instance
- ncomp: number of components. Should be lower than the number of columns of variates

Details
explained_variance calculates the explained variance of each variates out of the total variance in data.

Value
explained_variance simply returns the explained variance for each variate.

Author(s)
Florian Rohart

See Also
spls, splsda, plotIndiv, plotVar, cim, network.
get.confusion_matrix

Create confusion table and calculate the Balanced Error Rate

Description
Create confusion table between a vector of true classes and a vector of predicted classes, calculate the Balanced Error rate

Usage
get.confusion_matrix(truth, all.levels, predicted)
get.BER(confusion)

Arguments
truth A factor vector indicating the true classes of the samples (typically Y from the training set).
all.levels Levels of the 'truth' factor. Optional parameter if there are some missing levels in truth compared to the fitted predicted model
predicted Vector of predicted classes (typically the prediction from the test set). Can contain NA.
confusion result from a get.confusion_matrix to calculate the Balanced Error Rate

Details
BER is appropriate in case of an unbalanced number of samples per class as it calculates the average proportion of wrongly classified samples in each class, weighted by the number of samples in each class. BER is less biased towards majority classes during the performance assessment.
get.confusion_matrix

Value

get.confusion_matrix returns a confusion matrix.
get.BER returns the BER from a confusion matrix

Author(s)

Florian Rohart

References

mixOmics manuscript:

See Also

predict.

Examples

# Example
# ----------------------------------------
## Not run:

data(liver.toxicity)
X <- liver.toxicity$gene
Y <- as.factor(liver.toxicity$treatment[, 4])

## if training is performed on 4/5th of the original data
samp <- sample(1:5, nrow(X), replace = TRUE)
test <- which(samp == 1)  # testing on the first fold
train <- setdiff(1:nrow(X), test)

plsda.train <- plsda(X[train, ], Y[train], ncomp = 2)
test.predict <- predict(plsda.train, X[test, ], dist = "max.dist")
Prediction <- test.predict$class$max.dist[, 2]

# the confusion table compares the real subtypes with the predicted subtypes for a 2 component model
confusion.mat = get.confusion_matrix(truth = Y[test],
predicted = Prediction)

get.BER(confusion.mat)

## End(Not run)
Description

This function provides an image map (checkerboard plot) of the cross-validation score obtained by the `tune.rcc` function.

Usage

```r
## S3 method for class 'tune.rcc'
image(x, col = heat.colors, ...)
```

Arguments

- `x`: object returned by `estim.regul`.
- `col`: a character string specifying the colors function to use: `terrain.colors`, `topo.colors`, `rainbow` or similar functions. Defaults to `heat.colors`.
- `...`: not used currently.

Details

`image.estim.regul` creates an image map of the matrix object$mat containing the cross-validation score obtained by the `estim.regul` function. Also a color scales strip is plotted.

Author(s)

Sebastien Dejean and Ignacio Gonzalez.

See Also

tune.rcc, image.

Examples

```r
data(nutrimouse)
X <- nutrimouse$lipid
Y <- nutrimouse$gene

## this can take some seconds
cv.score <- tune.rcc(X, Y, validation = "Mfold", plot = FALSE)
image(cv.score)
```
image.estim.regul

Plot the cross-validation score.

Description

This function has been renamed 'image.tune.rcc', see image.tune.rcc.

imgCor

Image Maps of Correlation Matrices between two Data Sets

Description

Display two-dimensional visualizations (image maps) of the correlation matrices within and between two data sets.

Usage

imgCor(X, Y, type = "combine", X.var.names = TRUE, Y.var.names = TRUE, sideColors = TRUE, interactive.dev = TRUE, title = TRUE, color, row.cex, col.cex, symkey, keysize, xlab, ylab, margins, lhei, lwid)

Arguments

X numeric matrix or data frame (n x p), the observations on the X variables. NAs are allowed.

Y numeric matrix or data frame (n x q), the observations on the Y variables. NAs are allowed.

type character string, (partially) matching one of "combine" or "separated", determining the kind of plots to be produced. See Details.

X.var.names, Y.var.names logical, should the name of X- and/or Y-variables be shown? If TRUE (default) object$namesX and/or object$namesY are used. Possible character vector with X- and/or Y-variable labels to use.

sideColors character vector of length two. The color name for horizontal and vertical side bars that may be used to annotate the X and Y correlation matrices.

interactive.dev boolean. The current graphics device that will be opened is interactive?
imgCor

title logical, should the main titles be shown?
color, xlab, ylab arguments passed to cim.
row.cex, col.cex
positive numbers, used as cex.axis in for the row or column axis labeling. The
defaults currently only use number of rows or columns, respectively.
symkey boolean indicating whether the color key should be made symmetric about 0.
Defaults to TRUE.
keysiz e positive numeric value indicating the size of the color key.
margins numeric vector of length two containing the margins (see par(mar)) for column
and row names respectively.
lhei, lwid arguments passed to layout to divide the device up into two rows and two
columns, with the row-heights lhei and the column-widths lwid.

Details

If type="combine", the correlation matrix is computed of the combined matrices cbind(X, Y)
and then plotted. If type="separate", three correlation matrices are computed, cor(X), cor(Y)
and cor(X, Y) and plotted separately on a device. In both cases, a color correlation scales strip is
plotted.

The correlation matrices are pre-processed before calling the image function in order to get, as in
the numerical representation, the diagonal from upper-left corner to bottom-right one.

Missing values are handled by casewise deletion in the imgCor function.
If X.names = FALSE, the name of each X-variable is hidden. Default value is TRUE.
If Y.names = FALSE, the name of each Y-variable is hidden. Default value is TRUE.

Author(s)
Ignacio Gonzalez.

See Also
cor, image, color.jet.

Examples

data(nutrimouse)
X <- nutrimouse$lipid
Y <- nutrimouse$gene

### 'combine' type plot (default)
imgCor(X, Y)

### 'separate' type plot
### Not run:
imgCor(X, Y, type = "separate")
## ipca

**Independent Principal Component Analysis**

### Description

Performs independent principal component analysis on the given data matrix, a combination of Principal Component Analysis and Independent Component Analysis.

### Usage

```r
ipca(x, 
ncomp = 2, 
mode = "deflation", 
fun = "logcosh", 
scale = FALSE, 
w.init = NULL, 
max.iter = 200, 
tol = 1e-04)
```

### Arguments

- **X**
  a numeric matrix (or data frame) which provides the data for the principal component analysis.

- **ncomp**
  integer, number of independent component to choose. Set by default to 3.

- **mode**
  character string. What type of algorithm to use when estimating the unmixing matrix, choose one of "deflation", "parallel". Default set to deflation.

- **fun**
  the function used in approximation to neg-entropy in the FastICA algorithm. Default set to logcosh, see details of FastICA.

- **scale**
  a logical value indicating whether the variables (columns) of the data matrix X should be standardized beforehand. By default, X is centered.

- **max.iter**
  integer, maximum number of iterations to perform.

- **tol**
  a positive scalar giving the tolerance at which the un-mixing matrix is considered to have converged, see fastICA package.

- **w.init**
  initial un-mixing matrix (unlike FastICA, this matrix is fixed here).
**Details**

In PCA, the loading vectors indicate the importance of the variables in the principal components. In large biological data sets, the loading vectors should only assign large weights to important variables (genes, metabolites ...). That means the distribution of any loading vector should be super-Gaussian: most of the weights are very close to zero while only a few have large (absolute) values.

However, due to the existence of noise, the distribution of any loading vector is distorted and tends toward a Gaussian distribution according to the Central Limit Theorem. By maximizing the non-Gaussianity of the loading vectors using FastICA, we obtain more noiseless loading vectors. We then project the original data matrix on these noiseless loading vectors, to obtain independent principal components, which should be also more noiseless and be able to better cluster the samples according to the biological treatment (note, IPCA is an unsupervised approach).

**Algorithm**

1. The original data matrix is centered.
2. PCA is used to reduce dimension and generate the loading vectors.
3. ICA (FastICA) is implemented on the loading vectors to generate independent loading vectors.
4. The centered data matrix is projected on the independent loading vectors to obtain the independent principal components.

**Value**

ipca returns a list with class "ipca" containing the following components:

- `ncomp`: the number of independent principal components used.
- `unmixing`: the unmixing matrix of size (ncomp x ncomp)
- `mixing`: the mixing matrix of size (ncomp x ncomp)
- `x`: the centered data matrix
- `x`: the independent principal components
- `loadings`: the independent loading vectors
- `kurtosis`: the kurtosis measure of the independent loading vectors

**Author(s)**

Fangzhou Yao and Jeff Coquery.

**References**


**See Also**

sipca, pca, plotIndiv, plotVar, and http://www.mixOmics.org for more details.
Examples

```r
data(liver.toxicity)

# implement IPCA on a microarray dataset
ipca.res <- ipca(liver.toxicity$gene, ncomp = 3, mode="deflation")

# samples representation
plotIndiv(ipca.res, ind.names = as.character(liver.toxicity$treatment[, 4]),
          group = as.numeric(as.factor(liver.toxicity$treatment[, 4])))
## Not run:
plotIndiv(ipca.res, cex = 0.01,
          col = as.numeric(as.factor(liver.toxicity$treatment[, 4])),style="3d")
## End(Not run)

# variables representation
plotVar(ipca.res, cex = 0.5)

## Not run:
plotVar(ipca.res, rad.in = 0.5, cex = 0.5,style="3d")
## End(Not run)
```

Description

The 16S data come from Koren et al. (2011) and compared the bodysites oral, gut and plaque microbial communities in patients with atherosclerosis. The data can be analysed with our mixMC module. The data include 43 samples measured on 980 OTUs.

Usage

```r
data(Koren.16S)
```

Format

A list containing two data sets, `data.TSS` and `data.raw` and some meta data information:

- `data.TSS` data frame with 43 rows (samples) and 980 columns (OTUs). The prefiltered normalised data using Total Sum Scaling normalisation.
- `data.raw` data frame with 43 rows (samples) and 980 columns (OTUs). The prefiltered raw count OTU data which include a 1 offset (i.e. no 0 values).
- `taxonomy` data frame with 980 rows (OTUs) and 7 columns indicating the taxonomy of each OTU.
- `indiv` data frame with 43 rows indicating sample meta data.
- `bodysite` factor of length 43 indicating the bodysite with levels arterial plaque, saliva and stool.
Details

The data are from Koren et al. (2011) who examined the link between oral, gut and plaque microbial communities in patients with atherosclerosis and controls. Only healthy individuals were retained in the analysis. This study contained partially repeated measures from multiple sites including 15 unique patients samples from saliva and stool, and 13 unique patients only sampled from arterial plaque samples and we therefore considered a non multilevel analysis for that experimental design. After prefiltering, the data included 973 OTU for 43 samples. We strongly recommend to use log ratio transformations on the data. Normalised data, as implemented in the PLS and PCA methods, see details on www.mixOmics.org/mixMC.

The data.raw include a 1 offset in order to be log ratios transformed after TSS normalisation. Consequently, the data.TSS are TSS normalisation of data.raw. The CSS normalisation was performed on the original data (including zero values).

Source

The raw data were downloaded from the QIITA database. Filtering and normalisation described in our website www.mixOmics.org/mixMC

References


linnerud Dataset

Description

Three physiological and three exercise variables are measured on twenty middle-aged men in a fitness club.

Usage

data(linnerud)

Format

A list containing the following components:

- exercise  data frame with 20 observations on 3 exercise variables.
- physiological  data frame with 20 observations on 3 physiological variables.
liver.toxicity

Source

Tenenhaus, M. (1998), Table 1, page 15.

References


---

**Liver Toxicity Data**

**Description**

This data set contains the expression measure of 3116 genes and 10 clinical measurements for 64 subjects (rats) that were exposed to non-toxic, moderately toxic or severely toxic doses of acetaminophen in a controlled experiment.

**Usage**

data(liver.toxicity)

**Format**

A list containing the following components:

- `gene` data frame with 64 rows and 3116 columns. The expression measure of 3116 genes for the 64 subjects (rats).
- `clinic` data frame with 64 rows and 10 columns, containing 10 clinical variables for the same 64 subjects.
- `treatment` data frame with 64 rows and 4 columns, containing the treatment information on the 64 subjects, such as doses of acetaminophen and times of necropsies.
- `gene.ID` data frame with 3116 rows and 2 columns, containing geneBank IDs and gene titles of the annotated genes.

**Details**

The data come from a liver toxicity study (Bushel *et al.*, 2007) in which 64 male rats of the inbred strain Fisher 344 were exposed to non-toxic (50 or 150 mg/kg), moderately toxic (1500 mg/kg) or severely toxic (2000 mg/kg) doses of acetaminophen (paracetamol) in a controlled experiment. Necropsies were performed at 6, 18, 24 and 48 hours after exposure and the mRNA from the liver was extracted. Ten clinical chemistry measurements of variables containing markers for liver injury are available for each subject and the serum enzymes levels are measured numerically. The data were further normalized and pre-processed by Bushel *et al.* (2007).

**Source**

The two liver toxicity data sets are a companion resource for the paper of Bushel *et al.* (2007), and was downloaded from:

http://www.biomedcentral.com/1752-0509/1/15/additional/
References


logratio.transfo

Log-ratio transformation

Description

This function applies a log transformation to the data, either CLR or ILR

Usage

logratio.transfo(X, logratio = "none", offset = 0)

Arguments

X numeric matrix of predictors
logratio log-ratio transform to apply, one of "none", "CLR" or "ILR"
offset Value that is added to X for CLR and ILR log transformation. Default to 0.

Details

logratio.transfo applies a log transformation to the data, either CLR (centered log ratio transformation) or ILR (Isometric Log Ratio transformation). In the case of CLR log-transformation, X needs to be a matrix of non-negative values and offset is used to shift the values away from 0, as commonly done with counts data.

Value

logratio.transfo simply returns the log-ratio transformed data.

Author(s)

Florian Rohart
References

Kim-Anh Le Cao, Mary-Ellen Costello, Vanessa Anne Lakis, Francois Bartolo, Xin-Yi Chua, Remi Brazeilles, Pascale Rondeau mixMC: a multivariate statistical framework to gain insight into Microbial Communities bioRxiv 044206; doi: http://dx.doi.org/10.1101/044206


See Also

pca, pls, spls, plsda, splsda.

map

Classification given Probabilities

Description

Converts a matrix in which each row sums to 1 into the nearest matrix of (0,1) indicator variables.

Usage

map(y)

Arguments

y A matrix (for example a matrix of conditional probabilities in which each row sums to 1).

Value

A integer vector with one entry for each row of Y, in which the i-th value is the column index at which the i-th row of Y attains a maximum.

References


See Also

unmap
Examples

```r
data(nutrimouse)
Y = unmap(nutrimouse$diet)
map(Y)
```

---

**mat.rank**  
*Matrix Rank*

**Description**

This function estimates the rank of a matrix.

**Usage**

```r
mat.rank(mat, tol)
```

**Arguments**

- `mat`: a numeric matrix or data frame that can contain missing values.
- `tol`: positive real, the tolerance for singular values, only those with values larger than `tol` are considered non-zero.

**Details**

`mat.rank` estimate the rank of a matrix by computing its singular values $d[i]$ (using `nipals`). The rank of the matrix can be defined as the number of singular values $d[i] > 0$.

If `tol` is missing, it is given by $tol = max(dim(mat))*max(d)\times Machine\$double\_eps$.

**Value**

The returned value is a list with components:

- `rank`: an integer value, the matrix rank.
- `tol`: the tolerance used for singular values.

**Author(s)**

Sebastien Dejean and Ignacio Gonzalez.

**See Also**

`nipals`
Examples

```r
# Hilbert matrix
hilbert <- function(n) { i <- 1:n; 1 / outer(i - 1, i, "+") }
mat <- hilbert(16)
mat.rank(mat)

# Hilbert matrix with missing data
idx.na <- matrix(sample(c(0, 1, 1, 1), 36, replace = TRUE), ncol = 6)
m.na <- m <- hilbert(9)[, 1:6]
m.na[idx.na == 0] <- NA
mat.rank(m)
mat.rank(m.na)
```

Description

Function to integrate data sets measured on the same samples (N-integration) and to combine multiple independent studies measured on the same variables or predictors (P-integration) using variants of multi-group and generalised PLS (unsupervised analysis).

Usage

```r
mint.block.pls(X, Y, indY, study, ncomp = 2, design, scheme, mode, scale = TRUE, bias, init, tol = 1e-06, verbose, max.iter = 100, near.zero.var = FALSE)
```

Arguments

- `X` A list of data sets (called 'blocks') measured on the same samples. Data in the list should be arranged in samples x variables, with samples order matching in all data sets.
Y Matrix or vector response for a multivariate regression framework. Data should be continuous variables (see mint.block.plsda for supervised classification and factor response)

indy To be supplied if Y is missing, indicates the position of the matrix / vector response in the list X

study factor indicating the membership of each sample to each of the studies being combined

ncomp the number of components to include in the model. Default to 2.

design numeric matrix of size (number of blocks) x (number of blocks) with only 0 or 1 values. A value of 1 (0) indicates a relationship (no relationship) between the blocks to be modelled. If Y is provided instead of indY, the design matrix is changed to include relationships to Y.

scheme Either "horst", "factorial" or "centroid". Default = horst, see reference.

mode character string. What type of algorithm to use, (partially) matching one of "regression", "canonical", "invariant" or "classic". See Details.

scale booleean. If scale = TRUE, each block is standardized to zero means and unit variances (default: TRUE)

bias booleean. A logical value for biased or unbiased estimator of the var/cov (defaults to FALSE).

init Mode of initialization use in the algorithm, either by Singular Value Decomposition of the product of each block of X with Y ("svd") or each block independently ("svd.single"). Default = svd.single.

tol Convergence stopping value.

verbose if set to TRUE, reports progress on computing.

max.iter integer, the maximum number of iterations.

ear.zero.var boolean, see the internal nearZeroVar function (should be set to TRUE in particular for data with many zero values). Setting this argument to FALSE (when appropriate) will speed up the computations. Default value is FALSE.

Details

The function fits multi-group generalised PLS models with a specified number of ncomp components. An outcome needs to be provided, either by Y or by its position indY in the list of blocks X.

Multi (continuous)response are supported. X and Y can contain missing values. Missing values are handled by being disregarded during the cross product computations in the algorithm block.pls without having to delete rows with missing data. Alternatively, missing data can be imputed prior using the nipals function.

The type of algorithm to use is specified with the mode argument. Four PLS algorithms are available: PLS regression ("regression"), PLS canonical analysis ("canonical"), redundancy analysis ("invariant") and the classical PLS algorithm ("classic") (see References and more details in ?pls).
**Value**

`mint.block.pls` returns an object of class "mint.pls", "block.pls", a list that contains the following components:

- **X**: the centered and standardized original predictor matrix.
- **Y**: the centered and standardized original response vector or matrix.
- **ncomp**: the number of components included in the model for each block.
- **mode**: the algorithm used to fit the model.
- **mat.c**: matrix of coefficients from the regression of X / residual matrices X on the X-variates, to be used internally by `predict`.
- **variates**: list containing the X and Y variates.
- **loadings**: list containing the estimated loadings for the variates.
- **names**: list containing the names to be used for individuals and variables.
- **nzv**: list containing the zero- or near-zero predictors information.
- **tol**: the tolerance used in the iterative algorithm, used for subsequent S3 methods
- **max.iter**: the maximum number of iterations, used for subsequent S3 methods
- **iter**: Number of iterations of the algorithm for each component

**Author(s)**

Florian Rohart, Benoit Gautier, Kim-Anh Le Cao

**References**


**See Also**


**Examples**

# we will soon provide more examples on our website (data too large to be included
# in the package and still in active development)
Description

Function to integrate data sets measured on the same samples (N-integration) and to combine multiple independent studies measured on the same variables or predictors (P-integration) using variants of multi-group and generalised PLS-DA for supervised classification.

Usage

mint.block.plsda(x, y, indY, study, ncomp = 2, design, scheme, mode, scale = TRUE, bias, init, tol = 1e-06, verbose, max.iter = 100, near.zero.var = FALSE)

Arguments

X A list of data sets (called 'blocks') measured on the same samples. Data in the list should be arranged in samples x variables, with samples order matching in all data sets.

Y A factor or a class vector indicating the discrete outcome of each sample.

indY To be supplied if Y is missing, indicates the position of the matrix / vector response in the list X

study factor indicating the membership of each sample to each of the studies being combined

comp Number of components to include in the model (see Details). Default to 2.

design numeric matrix of size (number of blocks in X) x (number of blocks in X) with 0 or 1 values. A value of 1 (0) indicates a relationship (no relationship) between the blocks to be modelled. If Y is provided instead of indY, the design matrix is changed to include relationships to Y.

scheme Either "horst", "factorial" or "centroid". Default = horst, see reference.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>mode</strong></td>
<td>character string. What type of algorithm to use, (partially) matching one of</td>
</tr>
<tr>
<td></td>
<td>&quot;regression&quot;, &quot;canonical&quot;, &quot;invariant&quot; or &quot;classic&quot;. See Details.</td>
</tr>
<tr>
<td><strong>scale</strong></td>
<td>boolean. If scale = TRUE, each block is standardized to zero means and unit</td>
</tr>
<tr>
<td></td>
<td>variances (default: TRUE)</td>
</tr>
<tr>
<td><strong>bias</strong></td>
<td>boolean. A logical value for biased or unbiased estimator of the var/cov (de-</td>
</tr>
<tr>
<td></td>
<td>faults to FALSE)</td>
</tr>
<tr>
<td><strong>init</strong></td>
<td>Mode of initialization use in the algorithm, either by Singular Value Decom-</td>
</tr>
<tr>
<td></td>
<td>position of the product of each block of X with Y (&quot;svd&quot;) or each block in-</td>
</tr>
<tr>
<td></td>
<td>dependently (&quot;svd.single&quot;). Default = svd.single.</td>
</tr>
<tr>
<td><strong>tol</strong></td>
<td>Convergence stopping value.</td>
</tr>
<tr>
<td><strong>verbose</strong></td>
<td>if set to TRUE, reports progress on computing.</td>
</tr>
<tr>
<td><strong>max.iter</strong></td>
<td>integer, the maximum number of iterations.</td>
</tr>
<tr>
<td><strong>near.zero.var</strong></td>
<td>boolean, see the internal nearZeroVar function (should be set to TRUE in</td>
</tr>
<tr>
<td></td>
<td>particular for data with many zero values). Setting this argument to FALSE</td>
</tr>
<tr>
<td></td>
<td>(when appropriate) will speed up the computations. Default value is FALSE</td>
</tr>
</tbody>
</table>

**Details**

The function fits multi-group generalised PLS models with a specified number of ncomp components. A factor indicating the discrete outcome needs to be provided, either by Y or by its position indY in the list of blocks X.

X can contain missing values. Missing values are handled by being disregarded during the cross product computations in the algorithm block.pls without having to delete rows with missing data. Alternatively, missing data can be imputed prior using the nipals function.

The type of algorithm to use is specified with the mode argument. Four PLS algorithms are available: PLS regression ("regression"), PLS canonical analysis ("canonical"), redundancy analysis ("invariant") and the classical PLS algorithm ("classic") (see References and more details in ?pls).

**Value**

mint.block.plsda returns an object of class "mint.plsda", "block.plsda", a list that contains the following components:

- **X** the centered and standardized original predictor matrix.
- **Y** the centered and standardized original response vector or matrix.
- **ncomp** the number of components included in the model for each block.
- **mode** the algorithm used to fit the model.
- **mat.c** matrix of coefficients from the regression of X / residual matrices X on the X-variates, to be used internally by predict.
- **variates** list containing the X and Y variates.
- **loadings** list containing the estimated loadings for the variates.
- **names** list containing the names to be used for individuals and variables.
- **nzv** list containing the zero- or near-zero predictors information.
tol  the tolerance used in the iterative algorithm, used for subsequent S3 methods
max.iter the maximum number of iterations, used for subsequent S3 methods
iter  Number of iterations of the algorithm for each component

Author(s)
Florian Rohart, Benoit Gautier, Kim-Anh Le Cao

References

On multi-group PLS:
tive approach to identify a reproducible biomarker signature across multiple experiments and

J. Chemometrics, 28(3), 192-201.

On multiple integration with PLSDA:
org/content/early/2016/08/03/067611 Tenenhaus A., Philippe C., Guillemot V, Le Cao K.A.,
kxu001

Gunther O., Shin H., Ng R. T., McMaster W. R., McManus B. M., Keown P. A., Tebbutt S.J. ,
data: Applications in a kidney transplant rejection study, OMICS: A journal of integrative biology,
18(11), 682-95.

mixOomics manuscript:
and multiple data integration. BioRxiv available here: http://biorxiv.org/content/early/
2017/02/14/108597

See Also
spls, summary, plotIndiv, plotVar, predict, perf, mint.block.spls, mint.block.plsda,
mint.block.splsd and http://www.mixOomics.org/mixMINT for more details.

Examples
# we will soon provide more examples on our website (data too large to be included in the package
# and still in active development)
Description

Function to integrate data sets measured on the same samples (N-integration) and to combine multiple independent studies (P-integration) using variants of sparse multi-group and generalised PLS with variable selection (unsupervised analysis).

Usage

mint.block.spls(X, Y, indY, study, ncomp = 2, keepX.constraint, keepY.constraint, keepX, keepY, design, scheme, mode, scale = TRUE, bias, init, tol = 1e-06, verbose, max.iter = 100, near.zero.var = FALSE)

Arguments

X  A list of data sets (called 'blocks') measured on the same samples. Data in the list should be arranged in samples x variables, with samples order matching in all data sets.

Y  Matrix or vector response for a multivariate regression framework. Data should be continuous variables (see block.splsda for supervised classification and factor reponse)

indY  To supply if Y is missing, indicates the position of the matrix / vector response in the list X

study  factor indicating the membership of each sample to each of the studies being combined

ncomp  the number of components to include in the model. Default to 2.
keepX.constraint
A list of same length as X. Each entry indicates specific variables to be selected in each of the blocks of X for the first PLS-components, see details.

keepY.constraint
Only if Y is provided, otherwise extracted from keepX.constraint (see details). Each entry indicates specific variables to be selected in Y for the first PLS-components, see details.

keepX
A list of same length as X. Each entry is the number of variables to select in each of the blocks of X for each component. In the case where keepX.constraint is specified it should indicate the number of variables to select on the remaining components. By default all variables are kept in the model.

keepY
Only if Y is provided. Each entry is the number of variables to select in each of the blocks of Y for each component. In the case where keepX.constraint is specified it should indicate the number of variables to select on the remaining components. By default all variables are kept in the model.

design
numeric matrix of size (number of blocks in X) x (number of blocks in X) with 0 or 1 values. A value of 1 (0) indicates a relationship (no relationship) between the blocks to be modelled. If Y is provided instead of indY, the design matrix is changed to include relationships to Y.

scheme
Either "horst", "factorial" or "centroid". Default = horst, see reference.

mode
character string. What type of algorithm to use, (partially) matching one of "regression", "canonical", "invariant" or "classic". See Details.

scale
boolean. If scale = TRUE, each block is standardized to zero means and unit variances (default: TRUE)

bias
boolean. A logical value for biased or unbiased estimator of the var/cov (defaults to FALSE).

init
Mode of initialization use in the algorithm, either by Singular Value Decomposition of the product of each block of X with Y ("svd") or each block independently ("svd.single"). Default = svd.single.

tol
Convergence stopping value.

verbose
if set to TRUE, reports progress on computing.

max.iter
integer, the maximum number of iterations.

near.zero.var
boolean, see the internal nearZeroVar function (should be set to TRUE in particular for data with many zero values). Setting this argument to FALSE (when appropriate) will speed up the computations. Default value is FALSE

Details
The function fits sparse multi-group generalised PLS models with a specified number of ncomp components. An outcome needs to be provided, either by Y or by its position indY in the list of blocks X.

Multi (continuous)response are supported. X and Y can contain missing values. Missing values are handled by being disregarded during the cross product computations in the algorithm block.pls without having to delete rows with missing data. Alternatively, missing data can be imputed prior using the nipals function.
The type of algorithm to use is specified with the `mode` argument. Four PLS algorithms are available: PLS regression ("regression"), PLS canonical analysis ("canonical"), redundancy analysis ("invariant") and the classical PLS algorithm ("classic") (see References and more details in `?pls`).

Constraint model (`constraint=TRUE`). The model is run on the optimal list of selected features `keepX.constraint` from the previous components, as opposed to considering only a specified number of features `keepX`.

**Value**

`mint.block.spls` returns an object of class "mint.spls", "block.spls", a list that contains the following components:

- `X`: the centered and standardized original predictor matrix.
- `Y`: the centered and standardized original response vector or matrix.
- `ncomp`: the number of components included in the model for each block.
- `mode`: the algorithm used to fit the model.
- `mat.c`: matrix of coefficients from the regression of `X` / residual matrices `X` on the `X`-variates, to be used internally by `predict`.
- `variates`: list containing the `X` and `Y` variates.
- `loadings`: list containing the estimated loadings for the variates.
- `names`: list containing the names to be used for individuals and variables.
- `nzv`: list containing the zero- or near-zero predictors information.
- `tol`: the tolerance used in the iterative algorithm, used for subsequent S3 methods.
- `max.iter`: the maximum number of iterations, used for subsequent S3 methods.
- `iter`: Number of iterations of the algorithm for each component.

**Author(s)**

Florian Rohart, Benoit Gautier, Kim-Anh Le Cao

**References**


**See Also**


**Examples**

# we will soon provide more examples on our website (data too large to be included in the package
# and still in active development)
mint.block.splsda

**Description**

Function to integrate data sets measured on the same samples (N-integration) and to combine multiple independent studies measured on the same variables or predictors (P-integration) using variants of sparse multi-group and generalised PLS-DA for supervised classification and variable selection.

**Usage**

```r
mint.block.splsda(X, Y, indY, study, ncomp = 2, keepX.constraint, keepX, design, scheme, mode, scale = TRUE, bias, init, tol = 1e-06, verbose, max.iter = 100, near.zero.var = FALSE)
```

**Arguments**

- `X` A list of data sets (called ‘blocks’) measured on the same samples. Data in the list should be arranged in samples x variables, with samples order matching in all data sets.
- `Y` A factor or a class vector indicating the discrete outcome of each sample.
- `indY` To be supplied if Y is missing, indicates the position of the matrix / vector response in the list X
- `study` factor indicating the membership of each sample to each of the studies being combined
- `ncomp` Number of components to include in the model (see Details). Default to 2.
- `keepX.constraint` A list of same length as X. Each entry indicates specific variables to be selected in each of the blocks of X for the first PLS-components.
### mint.block.splsda

- **keepX**: A list of same length as X. Each entry is the number of variables to select in each of the blocks of X for each component. In the case where `keepX.constraint` is specified it should indicate the number of variables to select on the remaining components. By default all variables are kept in the model.

- **design**: Numeric matrix of size (number of blocks in X) x (number of blocks in X) with 0 or 1 values. A value of 1 (0) indicates a relationship (no relationship) between the blocks to be modelled. If Y is provided instead of `indy`, the design matrix is changed to include relationships to Y.

- **scheme**: Either "horst", "factorial" or "centroid". Default = horst, see reference.

- **mode**: Character string. What type of algorithm to use, (partially) matching one of "regression", "canonical", "invariant" or "classic". See Details.

- **scale**: Boolean. If scale = TRUE, each block is standardized to zero means and unit variances (default: TRUE)

- **bias**: Boolean. A logical value for biased or unbiased estimator of the var/cov (defaults to FALSE).

- **init**: Mode of initialization use in the algorithm, either by Singular Value Decomposition of the product of each block of X with Y ("svd") or each block independently ("svd.single"). Default = svd.single.

- **tol**: Convergence stopping value.

- **verbose**: If set to TRUE, reports progress on computing.

- **max.iter**: Integer, the maximum number of iterations.

- **near.zero.var**: Boolean, see the internal nearZeroVar function (should be set to TRUE in particular for data with many zero values). Setting this argument to FALSE (when appropriate) will speed up the computations. Default value is FALSE

### Details

The function fits sparse multi-group generalised PLS Discriminant Analysis models with a specified number of ncomp components. A factor indicating the discrete outcome needs to be provided, either by `Y` or by its position `indy` in the list of blocks `X`.

X can contain missing values. Missing values are handled by being disregarded during the cross product computations in the algorithm block.pls without having to delete rows with missing data. Alternatively, missing data can be imputed prior using the nipals function.

The type of algorithm to use is specified with the `mode` argument. Four PLS algorithms are available: PLS regression ("regression"), PLS canonical analysis ("canonical"), redundancy analysis ("invariant") and the classical PLS algorithm ("classic") (see References and more details in ?pls).

Constraint model. To improve performance, a new argument was recently added to fit a constraint model (constraint=TRUE). The model is run on the optimal list of selected features `keepX.constraint` from the previous components, as opposed to considering only a specified number of features `keepX`. Such strategy was implemented in the sister package bootPLS and successfully applied in our recent integrative study (Rohart et al., 2016). Our experience has shown that the constraint model can substantially improve the performance of the method.)
Value

`mint.block.splsda` returns an object of class "`mint.splsda`", "`block.splsda`", a list that contains the following components:

- **X**
  the centered and standardized original predictor matrix.

- **Y**
  the centered and standardized original response vector or matrix.

- **ncomp**
  the number of components included in the model for each block.

- **mode**
  the algorithm used to fit the model.

- **mat.c**
  matrix of coefficients from the regression of X / residual matrices X on the X-variates, to be used internally by `predict`.

- **variates**
  list containing the X and Y variates.

- **loadings**
  list containing the estimated loadings for the variates.

- **names**
  list containing the names to be used for individuals and variables.

- **nzv**
  list containing the zero- or near-zero predictors information.

- **tol**
  the tolerance used in the iterative algorithm, used for subsequent S3 methods

- **max.iter**
  the maximum number of iterations, used for subsequent S3 methods

- **iter**
  Number of iterations of the algorithm for each component

Author(s)

Florian Rohart, Benoit Gautier, Kim-Anh Le Cao

References


mixOmics manuscript:

mint.pca

See Also


Examples

# we will soon provide more examples on our website (data too large to be included in the package
# and still in active development)

mint.pca  P-integration with Principal Component Analysis

Description

Function to integrate and combine multiple independent studies measured on the same variables or predictors (P-integration) using a multigroup Principal Component Analysis.

Usage

mint.pca(X,
  ncomp = 2,
  study,
  scale = TRUE,
  tol = 1e-06,
  max.iter = 100
)

Arguments

X numeric matrix of predictors combining multiple independent studies on the same set of predictors. NAs are allowed.

ncomp Number of components to include in the model (see Details). Default to 2

study factor indicating the membership of each sample to each of the studies being combined

scale boolean. If scale = TRUE, each block is standardized to zero means and unit variances. Default = TRUE.

tol Convergence stopping value.

max.iter integer, the maximum number of iterations.
Details

mint.pca fits a vertical PCA model with ncomp components in which several independent studies measured on the same variables are integrated. The study factor indicates the membership of each sample in each study. We advise to only combine studies with more than 3 samples as the function performs internal scaling per study.

Missing values are handled by being disregarded during the cross product computations in the algorithm without having to delete rows with missing data. Alternatively, missing data can be imputed prior using the nipals function.

Useful graphical outputs are available, e.g. plotIndiv, plotLoadings, plotVar.

Value

mint.pca returns an object of class "mint.pca", "pca", a list that contains the following components:

- **X**: the centered and standardized original predictor matrix.
- **ncomp**: the number of components included in the model.
- **study**: The study grouping factor
- **sdev**: the eigenvalues of the covariance/correlation matrix, though the calculation is actually done with the singular values of the data matrix or by using NIPALS.
- **center, scale**: the centering and scaling used, or FALSE.
- **rotation**: the matrix of variable loadings (i.e., a matrix whose columns contain the eigenvectors).
- **loadings**: same as 'rotation' to keep the mixOmics spirit
- **x**: the value of the rotated data (the centred (and scaled if requested) data multiplied by the rotation/loadings matrix), also called the principal components.
- **variates**: same as 'x' to keep the mixOmics spirit
- **explained_variance**: explained variance from the multivariate model, used for plotIndiv
- **names**: list containing the names to be used for individuals and variables.

Author(s)

Florian Rohart, Kim-Anh Le Cao

References


mint.pls

See Also


Examples

data(stemcells)

res = mint.pls(X = stemcells$gene, ncomp = 3, study = stemcells$study)

plotIndiv(res, group = stemcells$celltype, legend=TRUE)

Description

Function to integrate and combine multiple independent studies measured on the same variables or predictors (P-integration) using variants of multi-group PLS (unsupervised analysis).

Usage

mint.pls(X,
Y,
ncomp = 2,
mode = c("regression", "canonical", "invariant", "classic"),
study,
scale = TRUE,
tol = 1e-06,
max.iter = 100,
near.zero.var = FALSE)

Arguments

X numeric matrix of predictors combining multiple independent studies on the same set of predictors. NAs are allowed.

Y Matrix or vector response for a multivariate regression framework. Data should be continuous variables (see mint.plsda for supervised classification and factor response)

ncomp Number of components to include in the model (see Details). Default to 2

mode character string. What type of algorithm to use, (partially) matching one of "regression" or "canonical". See Details.

study factor indicating the membership of each sample to each of the studies being combined
scale

booleean. If scale = TRUE, each block is standardized to zero means and unit variances. Default = TRUE.

tol

Convergence stopping value.

max.iter

integer, the maximum number of iterations.

near.zero.var

boolean, see the internal nearZeroVar function (should be set to TRUE in particular for data with many zero values). Default = FALSE.

Details

mint.pls fits a vertical PLS-DA models with ncomp components in which several independent studies measured on the same variables are integrated. The aim is to explain the continuous outcome y. The study factor indicates the membership of each sample in each study. We advise to only combine studies with more than 3 samples as the function performs internal scaling per study.

Multi (continuous)response are supported. X and Y can contain missing values. Missing values are handled by being disregarded during the cross product computations in the algorithm mint.pls without having to delete rows with missing data. Alternatively, missing data can be imputed prior using the nipals function.

The type of algorithm to use is specified with the mode argument. Four PLS algorithms are available: PLS regression ("regression"), PLS canonical analysis ("canonical"), redundancy analysis ("invariant") and the classical PLS algorithm ("classic") (see References and more details in ?pls).

Useful graphical outputs are available, e.g. plotIndiv, plotLoadings, plotVar.

Value

mint.pls returns an object of class "mint.pls", "pls", a list that contains the following components:

X

the centered and standardized original predictor matrix.

Y

the centered and standardized original response vector or matrix.

ncomp

the number of components included in the model.

study

The study grouping factor

mode

the algorithm used to fit the model.

variates

list containing the variates of X - global variates.

loadings

list containing the estimated loadings for the variates - global loadings.

variates.partial

list containing the variates of X relative to each study - partial variates.

loadings.partial

list containing the estimated loadings for the partial variates - partial loadings.

names

list containing the names to be used for individuals and variables.

nzv

list containing the zero- or near-zero predictors information.

iter

Number of iterations of the algorithm for each component

explained_variance

Percentage of explained variance for each component and each study (note that contrary to PCA, this amount may not decrease as the aim of the method is not to maximise the variance, but the covariance between data sets).
Author(s)
Florian Rohart, Kim-Anh Le Cao

References


See Also

Examples
# we will soon provide more examples on our website (data too large to be included in the package)

mint.plsda

P-integration with Projection to Latent Structures models (PLS) with Discriminant Analysis

Description
Function to combine multiple independent studies measured on the same variables or predictors (P-integration) using variants of multi-group PLS-DA for supervised classification.

Usage
mint.plsda(X, 
Y, 
ncomp = 2, 
mode = c("regression", "canonical", "invariant", "classic"), 
study, 
scale = TRUE, 
tol = 1e-06, 
max.iter = 100, 
near.zero.var = FALSE)
**Arguments**

- **X**
  numeric matrix of predictors combining multiple independent studies on the same set of predictors. NAs are allowed.

- **Y**
  A factor or a class vector indicating the discrete outcome of each sample.

- **ncomp**
  Number of components to include in the model (see Details). Default to 2

- **mode**
  character string. What type of algorithm to use, (partially) matching one of "regression" or "canonical". See Details.

- **study**
  factor indicating the membership of each sample to each of the studies being combined

- **scale**
  boolean. If scale = TRUE, each block is standardized to zero means and unit variances. Default = TRUE.

- **tol**
  Convergence stopping value.

- **max.iter**
  integer, the maximum number of iterations.

- **near.zero.var**
  boolean, see the internal `nearZeroVar` function (should be set to TRUE in particular for data with many zero values). Default = FALSE.

**Details**

`mint.plsda` function fits a vertical PLS-DA models with `ncomp` components in which several independent studies measured on the same variables are integrated. The aim is to classify the discrete outcome `Y`. The study factor indicates the membership of each sample in each study. We advise to only combine studies with more than 3 samples as the function performs internal scaling per study, and where all outcome categories are represented.

`X` can contain missing values. Missing values are handled by being disregarded during the cross product computations in the algorithm `mint.plsda` without having to delete rows with missing data. Alternatively, missing data can be imputed prior using the `nipals` function.

The type of algorithm to use is specified with the `mode` argument. Four PLS algorithms are available: PLS regression ("regression"), PLS canonical analysis ("canonical"), redundancy analysis ("invariant") and the classical PLS algorithm ("classic") (see References and more details in `_pls`).

Useful graphical outputs are available, e.g. `plotIndiv`, `plotLoadings`, `plotVar`.

**Value**

`mint.plsda` returns an object of class "`mint.plsda"", "plsda". a list that contains the following components:

- **X**
  the centered and standardized original predictor matrix.

- **Y**
  original factor

- **ind.mat**
  the centered and standardized original response vector or matrix.

- **ncomp**
  the number of components included in the model.

- **study**
  The study grouping factor

- **mode**
  the algorithm used to fit the model.
variants  list containing the variates of X - global variates.
loadings  list containing the estimated loadings for the variates - global loadings.
variants.partial  list containing the variates of X relative to each study - partial variates.
loadings.partial  list containing the estimated loadings for the partial variates - partial loadings.
names  list containing the names to be used for individuals and variables.
nzv  list containing the zero- or near-zero predictors information.
iter  Number of iterations of the algorithm for each component
explained_variance  Percentage of explained variance for each component and each study (note that contrary to PCA, this amount may not decrease as the aim of the method is not to maximise the variance, but the covariance between X and the dummy matrix Y).

Author(s)
Florian Rohart, Kim-Anh Le Cao

References
mixOmics manuscript:

See Also
spls, summary, plotIndiv, plotVar, predict, perf, mint.pls, mint.spls, mint.splsda and http://www.mixOmics.org/mixMINT for more details.

Examples
data(stemcells)

res = mint.plsda(X = stemcells$gene, Y = stemcells$celltype, ncomp = 3, study = stemcells$study)
plotIndiv(res)

#plot study-specific outputs for all studies
plotIndiv(res, study = "all.partial")
mint.spls

P-integration with variable selection

Description

Function to integrate and combine multiple independent studies measured on the same variables or predictors (P-integration) using variants of multi-group sparse PLS for variable selection (unsupervised analysis).

Usage

mint.spls(X, Y, ncomp = 2, mode = c("regression", "canonical", "invariant", "classic"), study, keepX.constraint = NULL, keepY.constraint = NULL, keepX = rep(ncol(X), ncomp), keepY = rep(ncol(Y), ncomp), scale = TRUE, tol = 1e-06, max.iter = 100, near.zero.var = FALSE)

Arguments

X numeric matrix of predictors combining multiple independent studies on the same set of predictors. NAs are allowed.

Y Matrix or vector response for a multivariate regression framework. Data should be continuous variables (see mint.splsda for supervised classification and factor response)

ncomp Number of components to include in the model. Default to 2

mode character string. What type of algorithm to use, (partially) matching one of "regression", "canonical", "invariant" or "classic". See Details.

study grouping factor indicating which samples are from the same study

keepX.constraint A list of vectors. Each vector entry indicates specific variables to be selected in X for each previous PLS component. See details.
**mint.spls**

keePY.constraint
A list of vectors. Each vector entry indicates specific variables to be selected in Y for each previous PLS component. See details.

keePX
numeric vector indicating the number of variables to select in X on each component, or remaining components should keepX.constraint be provided. By default all variables are kept in the model.

keePY
numeric vector indicating the number of variables to select in Y on each component, or remaining components should keepY.constraint be provided. By default all variables are kept in the model.

scale
boolean. If scale = TRUE, each block is standardized to zero means and unit variances. Default = TRUE.

tol
Convergence stopping value.

max.iter
integer, the maximum number of iterations.

near.zero.var
boolean, see the internal nearZeroVar function (should be set to TRUE in particular for data with many zero values). Default = FALSE.

**Details**

mint.spls fits a vertical sparse PLS-DA models with ncomp components in which several independent studies measured on the same variables are integrated. The aim is to explain the continuous outcome Y and selecting correlated features between both data sets X and Y. The study factor indicates the membership of each sample in each study. We advise to only combine studies with more than 3 samples as the function performs internal scaling per study.

Multi (continuous)response are supported. X and Y can contain missing values. Missing values are handled by being disregarded during the cross product computations in the algorithm mint.spls without having to delete rows with missing data. Alternatively, missing data can be imputed prior using the nipals function.

The type of algorithm to use is specified with the mode argument. Four PLS algorithms are available: PLS regression ("regression"), PLS canonical analysis ("canonical"), redundancy analysis ("invariant") and the classical PLS algorithm ("classic") (see References and more details in ?pls).

Constraint model (constraint=TRUE). The model is run on the optimal list of selected features keepX.constraint from the previous components, as opposed to considering only a specified number of features keepX.

Variable selection is performed on each component for each block of X, and for Y if specified, via input parameter keepX and keepY, and keepX.constraint and keepY.constraint for a constraint model. keepX.constraint and keepY.constraint specify variable names to be selected in the previous components of the model.

Note that both keepX.constraint and keepY.constraint must have same length.

Useful graphical outputs are available, e.g. plotIndiv, plotLoadings, plotVar.

**Value**

mint.spls returns an object of class "mint.spls", "spls", a list that contains the following components:
X  numeric matrix of predictors combining multiple independent studies on the same set of predictors. NAs are allowed.
Y  the centered and standardized original response vector or matrix.
ncomp  the number of components included in the model.
study  The study grouping factor
mode  the algorithm used to fit the model.
keepX  Number of variables used to build each component of X
keepY  Number of variables used to build each component of Y
keepX.constraint  list indicating which variables where constrained to be used to build the first component of X
keepY.constraint  list indicating which variables where constrained to be used to build the first component of Y
variates  list containing the variates of X - global variates.
loadings  list containing the estimated loadings for the variates - global loadings.
variates.partial  list containing the variates of X relative to each study - partial variates.
loadings.partial  list containing the estimated loadings for the partial variates - partial loadings.
names  list containing the names to be used for individuals and variables.
nzv  list containing the zero- or near-zero predictors information.
iter  Number of iterations of the algorithm for each component
explained_variance  Percentage of explained variance for each component and each study (note that contrary to PCA, this amount may not decrease as the aim of the method is not to maximise the variance, but the covariance between data sets).

Author(s)
Florian Rohart, Kim-Anh Le Cao

References

See Also
mint.splsda

Examples

# we will soon provide more examples on our website (data too large to be included in the package)

---

mint.splsda  

P-integration with Discriminant Analysis and variable selection

Description

Function to combine multiple independent studies measured on the same variables or predictors (P-integration) using variants of multi-group sparse PLS-DA for supervised classification with variable selection.

Usage

mint.splsda(X,
        Y,
        ncomp = 2,
        mode = c("regression", "canonical", "invariant", "classic"),
        study,
        keepX.constraint = NULL,
        keepX = rep(ncol(X), ncomp),
        scale = TRUE,
        tol = 1e-06,
        max.iter = 100,
        near.zero.var = FALSE)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>numeric matrix of predictors combining multiple independent studies on the same set of predictors. NAs are allowed.</td>
</tr>
<tr>
<td>Y</td>
<td>A factor or a class vector indicating the discrete outcome of each sample.</td>
</tr>
<tr>
<td>ncomp</td>
<td>Number of components to include in the model (see Details). Default to 2</td>
</tr>
<tr>
<td>mode</td>
<td>character string. What type of algorithm to use, (partially) matching one of &quot;regression&quot; or &quot;canonical&quot;. See Details.</td>
</tr>
<tr>
<td>study</td>
<td>factor indicating the membership of each sample to each of the studies being combined</td>
</tr>
<tr>
<td>keepX.constraint</td>
<td>A list of vectors. Each vector entry indicates specific variables to be selected in X for each previous PLS component. See details.</td>
</tr>
<tr>
<td>keepX</td>
<td>numeric vector indicating the number of variables to select in X on each component, or remaining components should keepX.constraint be provided. By default all variables are kept in the model.</td>
</tr>
<tr>
<td>scale</td>
<td>boolean. If scale = TRUE, each block is standardized to zero means and unit variances. Default = TRUE.</td>
</tr>
</tbody>
</table>
Details

mint.splsda function fits a vertical sparse PLS-DA models with ncomp components in which several independent studies measured on the same variables are integrated. The aim is to classify the discrete outcome $Y$ and select variables that explain the outcome. The study factor indicates the membership of each sample in each study. We advise to only combine studies with more than 3 samples as the function performs internal scaling per study, and where all outcome categories are represented.

$x$ can contain missing values. Missing values are handled by being disregarded during the cross product computations in the algorithm mint.splsda without having to delete rows with missing data. Alternatively, missing data can be imputed prior using the nipals function.

The type of algorithm to use is specified with the mode argument. Four PLS algorithms are available: PLS regression ("regression"), PLS canonical analysis ("canonical"), redundancy analysis ("invariant") and the classical PLS algorithm ("classic") (see References and more details in _pls).

Constraint model. To improve performance, a new argument was recently added to fit a constraint model (constraint=TRUE). The model is run on the optimal list of selected features keepX.constraint from the previous components, as opposed to considering only a specified number of features keepX. Such strategy was implemented in the sister package bootPLS and successfully applied in our recent integrative study (Rohart et al., 2016). Our experience has shown that the constraint model can substantially improve the performance of the method (see tune.mint.splsda).

Variable selection is performed on each component for $X$ via input parameter keepX and keepX.constraint afor a constraint model. keepX.constraint specifies variable names to be selected in the previous components of the model, see example.

Useful graphical outputs are available, e.g. plotIndiv, plotLoadings, plotVar.

Value

mint.splsda returns an object of class "mint.splsda", "splsda", a list that contains the following components:

- $X$ the centered and standardized original predictor matrix.
- $Y$ the centered and standardized original response vector or matrix.
- ind.mat the centered and standardized original response vector or matrix.
- ncomp the number of components included in the model.
- study The study grouping factor
- mode the algorithm used to fit the model.
- keepX Number of variables used to build each component of $X$
- keepX.constraint list indicating which variables where constrained to be used to build the first component of $X$
variates list containing the variates of X - global variates.
loadings list containing the estimated loadings for the variates - global loadings.
loadings.partial list containing the estimated loadings for the partial variates - partial loadings.
names list containing the names to be used for individuals and variables.
nzv list containing the zero- or near-zero predictors information.
iter Number of iterations of the algorithm for each component
explained_variance Percentage of explained variance for each component and each study (note that contrary to PCA, this amount may not decrease as the aim of the method is not to maximise the variance, but the covariance between X and the dummy matrix Y).

Author(s)
Florian Rohart, Kim-Anh Le Cao

References
mixOmics manuscript:

See Also

Examples
data(stemcells)

# -- feature selection with keepX
res = mint.splsda(X = stemcells$gene, Y = stemcells$celltype, ncomp = 3, keepX = c(10, 5, 15), study = stemcells$study)

plotIndiv(res)  
#plot study-specific outputs for all studies
plotIndiv(res, study = "all.partial")
# plot study-specific outputs for study "2"
plotIndiv(res, study = "2")

# -- feature selection with keepX.constraint and keepX
# we force mint.spls to build the PLS-component 1 and 2 with only specific genes
keepX.constraint = list(comp1 = c("ENSG00000138756", "ENSG00000101470"),
comp2 = c("ENSG00000120049", "ENSG00000138685", "ENSG000000204248", "ENSG000000103260"))

# we select 10 and 15 variables on component 3 and 4
res = mint.splsda(X = stemcells$gene, Y = stemcells$celltype, ncomp = 4, study = stemcells$study,
keepX.constraint = keepX.constraint, keepX = c(10, 15))

# plot study-specific outputs for study "2", "3" and "4"
plotIndiv(res, study = c(2, 3, 4))

# poor results as the genes constraint on comp 1 and 2 were not relevant

---

mixOmics  

**PLS-derived methods: one function to rule them all!**

**Description**

This function performs one of the PLS derived methods included in the mixOmics package that is the most appropriate for your input data, one of (mint).(block).(s)pls(da) depending on your input data (single data, list of data, discrete outcome, ...)

**Usage**

```
mixOmics(X, Y, indY, study, ncomp, keepX.constraint, keepY.constraint, keepX, keepY, design, tau = NULL,# rgcca, number between 0,1 or "optimal" scheme, mode, scale, bias, init, tol = 1e-06, verbose = FALSE, max.iter = 100, near.zero.var = FALSE)
```
Arguments

**X**
Input data. Either a matrix or a list of data sets (called 'blocks') matching on the same samples. Data should be arranged in samples x variables, with samples order matching in all data sets.

**Y**
Outcome. Either a numeric matrix of responses or a factor or a class vector for the discrete outcome.

**indy**
To supply if Y is missing, indicates the position of the outcome in the list X

**study**
grouping factor indicating which samples are from the same study

**ncomp**
If X is a data matrix, ncomp is a single value. If X is a list of data sets, ncomp is a numeric vector of length the number of blocks in X. The number of components to include in the model for each block (does not necessarily need to take the same value for each block).

**keepX.constraint**
A list containing which variables of X are to be kept on each of the first PLS-components

**keepY.constraint**
A list containing which variables of Y are to be kept on each of the first PLS-components

**keepX**
Number of variables to keep in the X-loadings

**keepY**
Number of variables to keep in the Y-loadings

**design**
numeric matrix of size (number of blocks) x (number of blocks) with only 0 or 1 values. A value of 1 (0) indicates a relationship (no relationship) between the blocks to be modelled. If Y is provided instead of indY, the design matrix is changed to include relationships to Y.

**tau**
numeric vector of length the number of blocks in X. Each regularization parameter will be applied on each block and takes the value between 0 (no regularization) and 1. If tau = "optimal" the shrinkage parameters are estimated for each block and each dimension using the Schafer and Strimmer (2005) analytical formula.

**scheme**
Either "horst", "factorial" or "centroid" (Default: "centroid"), see reference paper.

**mode**
character string. What type of algorithm to use, (partially) matching one of "regression", "canonical", "invariant" or "classic". See Details.

**scale**
boolean. If scale = TRUE, each block is standardized to zero means and unit variances (default: TRUE)

**bias**
boolean. A logical value for biaised or unbiaised estimator of the var/cov (defaults to FALSE).

**init**
Mode of initialization use in the algorithm, either by Singular Value Decomposition of the product of each block of X with Y ("svd") or each block independently ("svd.single"). Default to "svd".

**tol**
Convergence stopping value.

**verbose**
if set to TRUE, reports progress on computing.

**max.iter**
integer, the maximum number of iterations.
`near_zero.var` boolean, see the internal `nearZeroVar` function (should be set to TRUE in particular for data with many zero values). Setting this argument to FALSE (when appropriate) will speed up the computations. Default value is FALSE

**Details**

This function performs one of the PLS derived methods included in the mixOmics package that is the most appropriate for your input data, one of `(mint).(block).(s)pls(da)`. If your input data `x` is a matrix, then the algorithm is directed towards one of `(mint).(s)pls(da)` depending on your input data `y` (factor for the discrete outcome directs the algorithm to DA analysis) and whether you input a study parameter (MINT analysis) or a `keepX/keepX.constraint` parameter (sparse analysis).

If your input data `x` is a list of matrices, then the algorithm is directed towards one of `(mint).block.(s)pls(da)` depending on your input data `y` (factor for the discrete outcome directs the algorithm to DA analysis) and whether you input a study parameter (MINT analysis) or a `keepX/keepX.constraint` parameter (sparse analysis).

More details about the PLS modes in `?pls`.

**Author(s)**

Florian Rohart

**References**

mixOmics manuscript:

MINT models:


Integration of omics data sets:


Sparse SVD:

PLS-DA:

PLS:

On multilevel analysis:

Visualisations:

See Also

Examples

```r
## -- directed towards PLS framework because X is a matrix and the study argument is missing
# -----------------------------------------------
data(liver.toxicity)
X = liver.toxicity$gene
Y = liver.toxicity$clinic
Y.factor = as.factor(liver.toxicity$treatment[, 4])

# directed towards PLS
out = mixOmics(X, Y, ncomp = 2)

# directed towards sPLS because of keepX and/or keepY
out = mixOmics(X, Y, ncomp = 2, keepX = c(50, 50), keepY = c(10, 10))

# directed towards PLS-DA because Y is a factor
out = mixOmics(X, Y.factor, ncomp = 2)
```
# directed towards sPLS-DA because Y is a factor and there is a keepX
out = mixOmics(X, Y.factor, ncomp = 2, keepX = c(20, 20))

## -- directed towards block.pls framework because X is a list
# -----------------------------------------------
data(nutrimouse)
Y = unmap(nutrimouse$diet)
data = list(gene = nutrimouse$gene, lipid = nutrimouse$lipid, Y = Y)

# directed towards block PLS
out = mixOmics(X = data, Y = Y, ncomp = 3)

# directed towards block sPLS because of keepX and/or keepY
out = mixOmics(X = data, Y = Y, ncomp = 3, keepX = list(gene = c(10, 10), lipid = c(15, 15)))

# directed towards block PLS-DA because Y is a factor
out = mixOmics(X = data, Y = nutrimouse$diet, ncomp = 3)

# directed towards block sPLS-DA because Y is a factor and there is a keepX
out = mixOmics(X = data, Y = nutrimouse$diet, ncomp = 3, keepX = list(gene = c(10, 10), lipid = c(15, 15)))

## -- directed towards mint.pls framework because of the study factor
# -----------------------------------------------
data(stemcells)
# directed towards PLS
out = mixOmics(X = stemcells$gene, Y = unmap(stemcells$celltype), ncomp = 2)

# directed towards mint.PLS
out = mixOmics(X = stemcells$gene, Y = unmap(stemcells$celltype), ncomp = 2, study = stemcells$study)

# directed towards mint.sPLS because of keepX and/or keepY
out = mixOmics(X = stemcells$gene, Y = unmap(stemcells$celltype), ncomp = 2, study = stemcells$study, keepX = c(10, 5, 15))

# directed towards mint.PLS-DA because Y is a factor
out = mixOmics(X = stemcells$gene, Y = stemcells$celltype, ncomp = 2, study = stemcells$study)

# directed towards mint.sPLS-DA because Y is a factor and there is a keepX
out = mixOmics(X = stemcells$gene, Y = stemcells$celltype, ncomp = 2, study = stemcells$study, keepX = c(10, 5, 15))
**Description**

This data set contains the expression of 48 known human ABC transporters with patterns of drug activity in 60 diverse cancer cell lines (the NCI-60) used by the National Cancer Institute to screen for anticancer activity.

**Usage**

data(multidrug)

**Format**

A list containing the following components:

- **ABC.trans** data matrix with 60 rows and 48 columns. The expression of the 48 human ABC transporters.
- **compound** data matrix with 60 rows and 1429 columns. The activity of 1429 drugs for the 60 cell lines.
- **comp.name** character vector. The names or the NSC No. of the 1429 compounds.
- **cell.line** a list containing two character vector components: Sample the names of the 60 cell line which were analysed, and **Class** the phenotypes of the 60 cell lines.

**Details**

The data come from a pharmacogenomic study (Szakacs *et al.*, 2004) in which two kinds of measurements acquired on the NCI-60 cancer cell lines are considered:

- the expression of the 48 human ABC transporters measured by real-time quantitative RT-PCR for each cell line;
- the activity of 1429 drugs expressed as $G_{I_{50}}$ which corresponds to the concentration at which the drug induces 50% inhibition of cellular growth for the cell line tested.

The NCI-60 panel includes cell lines derived from cancers of colorectal (7 cell lines), renal(8), ovarian(6), breast(8), prostate(2), lung(9) and central nervous system origin(6), as well as leukemias(6) and melanomas(8). It was set up by the Developmental Therapeutics Program of the National Cancer Institute (NCI, one of the U.S. National Institutes of Health) to screen the toxicity of chemical compound repositories. The expressions of the 48 human ABC transporters is available as a supplement to the paper of Szakacs *et al.* (2004).

The drug dataset consiste of 118 compounds whose mechanisms of action are putatively classifiable (Weinstein *et al.*, 1992) and a larger set of 1400 compounds that have been tested multiple times and whose screening data met quality control criteria described elsewhere (Scherf *et al.*, 2000). The two were combined to form a joint dataset that included 1429 compounds.

**Source**

The NCI dataset was downloaded from The Genomics and Bioinformatics Group Supplemental Table S1 to the paper of Szakacs *et al.* (2004), [http://discover.nci.nih.gov/abc/2004_cancercell_abstract.jsp#supplement](http://discover.nci.nih.gov/abc/2004_cancercell_abstract.jsp#supplement)

References


nearZeroVar

Identification of zero- or near-zero variance predictors

Description

Borrowed from the *caret* package. It is used as an internal function in the PLS methods, but can also be used as an external function, in particular when the data contain a lot of zeroes values and need to be prefiletered beforehand.

This function diagnoses predictors that have one unique value (i.e. are zero variance predictors) or predictors that are have both of the following characteristics: they have very few unique values relative to the number of samples and the ratio of the frequency of the most common value to the frequency of the second most common value is large.

Usage

nearZeroVar(x, freqCut = 95/5, uniqueCut = 10)

Arguments

- `x`: a numeric vector or matrix, or a data frame with all numeric data.
- `freqCut`: the cutoff for the ratio of the most common value to the second most common value.
- `uniqueCut`: the cutoff for the percentage of distinct values out of the number of total samples.

Details

For example, an example of near zero variance predictor is one that, for 1000 samples, has two distinct values and 999 of them are a single value.

To be flagged, first the frequency of the most prevalent value over the second most frequent value (called the “frequency ratio”) must be above `freqCut`. Secondly, the “percent of unique values,” the number of unique values divided by the total number of samples (times 100), must also be below `uniqueCut`.

In the above example, the frequency ratio is 999 and the unique value percentage is 0.0001.
network

Value

nearZeroVar returns a list that contains the following components:

Position  a vector of integers corresponding to the column positions of the problematic predictors that will need to be removed.

Metrics  a data frame containing the zero- or near-zero predictors information with columns: freqRatio, the ratio of frequencies for the most common value over the second most common value and, percentUnique, the percentage of unique data points out of the total number of data points.

Author(s)

Max Kuhn, with speed improvements to nearZerVar by Allan Engelhardt; enhancements by Florian Rohart, and speed up improvements by Benoit Gautier for mixOmics

See Also

pls, spls, plsda, splsda

network  Relevance Network for (r)CCA and (s)PLS regression

Description

Display relevance associations network for (regularized) canonical correlation analysis and (sparse) PLS regression. The function avoids the intensive computation of Pearson correlation matrices on large data set by calculating instead a pair-wise similarity matrix directly obtained from the latent components of our integrative approaches (CCA, PLS, block.pls methods). The similarity value between a pair of variables is obtained by calculating the sum of the correlations between the original variables and each of the latent components of the model. The values in the similarity matrix can be seen as a robust approximation of the Pearson correlation (see Gonzalez et al. 2012 for a mathematical demonstration and exact formula). The advantage of relevance networks is their ability to simultaneously represent positive and negative correlations, which are missed by methods based on Euclidian distances or mutual information. Those networks are bipartite and thus only a link between two variables of different types can be represented. The network can be saved in a .glm format using the igraph package, the function write.graph and extracting the output object$gR, see details.

Usage

network(mat, comp = NULL, blocks = c(1,2), cutoff = NULL, row.names = TRUE, col.names = TRUE, block.var.names = TRUE,
color.node = NULL,
shape.node = NULL,
cex.node.name = 1,
color.edge = color.GreenRed(100),
lty.edge = "solid",
lwd.edge = 1,
show.edge.labels = FALSE,
cex.edge.label = 1,
show.color.key = TRUE,
symkey = TRUE,
keysize = c(1, 1),
keysize.label = 1,
b breaks,
interactive = FALSE,
layout.fun = NULL,
save = NULL,
name.save = NULL)

Arguments

mat numeric matrix of values to be represented.
comp atomic or vector of positive integers. The components to adequately account for the data association. Defaults to comp = 1.
cutoff numeric value between 0 and 1. The tuning threshold for the relevant associations network (see Details).
row.names, col.names character vector containing the names of X- and Y-variables.
color.node vector of length two, the colors of the X and Y nodes (see Details).
shape.node character vector of length two, the shape of the X and Y nodes (see Details).
color.edge vector of colors or character string specifying the colors function to using to color the edges, set to default to color.GreenRed(100) but other palettes can be chosen (see Details and Examples).
lty.edge character vector of length two, the line type for the edges (see Details).
lwd.edge vector of length two, the line width of the edges (see Details).
show.edge.labels logical. If TRUE, plot association values as edge labels (defaults to FALSE).
show.color.key boolean. If TRUE a color key should be plotted.
symkey boolean indicating whether the color key should be made symmetric about 0. Defaults to TRUE.
keysize numeric value indicating the size of the color key.
keysize.label vector of length 1, indicating the size of the labels and title of the color key.
b breaks (optional) either a numeric vector indicating the splitting points for binning mat into colors, or a integer number of break points to be used, in which case the break points will be spaced equally between min(mat) and max(mat).
interactive

logical. If TRUE, a scrollbar is created to change the cutoff value interactively.
(defaults to FALSE). See Details.

save

should the plot be saved? If so, argument to be set either to 'jpeg', 'tiff',
'png' or 'pdf'.

name.save

character string giving the name of the saved file.

cex.edge.label

the font size for the edge labels.

cex.node.name

the font size for the node labels.

blocks

a vector indicating the block variables to display.

block.var.names

either a list of vector components for variable names in each block or FALSE
for no names. If TRUE, the columns names of the blocks are used as names.

layout.fun

a function. It specifies how the vertices will be placed on the graph. See
help(layout) in the igraph package. Defaults to layout.fruchterman.reingold.

Details

network allows to infer large-scale association networks between the X and Y datasets in rcc or
spls. The output is a graph where each X- and Y-variable corresponds to a node and the edges
included in the graph portray associations between them.

In rcc, to identify X-Y pairs showing relevant associations, network calculate a similarity measure
between X and Y variables in a pair-wise manner: the scalar product value between every pairs of
vectors in dimension length(comp) representing the variables X and Y on the axis defined by Z_i
with i in comp, where Z_i is the equiangular vector between the i-th X and Y canonical variate.

In spls, if object$mode is regression, the similarity measure between X and Y variables is given
by the scalar product value between every pairs of vectors in dimension length(comp) representing
the variables X and Y on the axis defined by U_i with i in comp, where U_i is the i-th X variate.
If object$mode is canonical then X and Y are represented on the axis defined by U_i and V_i
respectively.

Variable pairs with a high similarity measure (in absolute value) are considered as relevant. By
changing the cutoff, one can tune the relevance of the associations to include or exclude relations-
ships in the network.

interactive=TRUE open two device, one for association network, one for scrollbar, and define an
interactive process: by clicking at each end ('-' or '+') of the scrollbar or at middle portion
of this. The position of the slider indicate which is the 'cutoff' value associated to the display
network.

The network can be saved in a .glm format using the igraph package, the function write.graph
and extracting the output object$gr.

The interactive process is terminated by clicking the second button and selecting ‘Stop’ from the
menu, or from the ‘Stop’ menu on the graphics window.

The color.node is a vector of length two, of any of the three kind of R colors, i.e., either a color
name (an element of colors()), a hexadecimal string of the form "#rrggbb", or an integer i
meaning palette()[i]. color.node[1] and color.node[2] give the color for filled nodes of the
X- and Y-variables respectively. Defaults to c("white", "white").

color.edge give the color to edges with colors corresponding to the values in mat. Defaults to
color.GreenRed(100) for negative (green) and positive (red) correlations. We also propose other
palettes of colors, such as `color.jet` and `color.spectral`, see help on those functions, and examples below. Other pallete of colors from the stats package can be used too.

`shape.node[1]` and `shape.node[2]` provide the shape of the nodes associate to $X$- and $Y$-variables respectively. Current acceptable values are "circle" and "rectangle". Defaults to `c("circle", "rectangle")`.

`lty.edge[1]` and `lty.edge[2]` give the line type to edges with positive and negative weight respectively. Can be one of "solid", "dashed", "dotted", "dotdash", "longdash" and "twodash". Defaults to `c("solid", "solid")`.

`lwd.edge[1]` and `lwd.edge[2]` provide the line width to edges with positive and negative weight respectively. This attribute is of type double with a default of `c(1, 1)`.

**Value**

`network` return a list containing the following components:

- `M` the correlation matrix used by `network`.
- `gR` a graph object to save the graph for cytoscape use (requires to load the `igraph` package).

**Warning**

If the number of variables is high, the generation of the network generation can take some time.

**Author(s)**

Ignacio Gonzalez and Kim-Anh Le Cao.

**References**


Relevance networks:


**See Also**

Examples

## network representation for objects of class 'rcc'
data(nutrimouse)
X <- nutrimouse$lipid
Y <- nutrimouse$gene
nutri.res <- rcc(X, Y, ncomp = 3, lambda1 = 0.064, lambda2 = 0.008)

## Not run:
# may not work on the Linux version, use Windows instead
# sometimes with Rstudio might not work because of margin issues,
# in that case save it as an image
jpeg('example1-network.jpeg', res = 600, width = 4000, height = 4000)
network(nutri.res, comp = 1:3, cutoff = 0.6)
dev.off()

## End(Not run)

## Changing the attributes of the network
## Not run:
# sometimes with Rstudio might not work because of margin issues,
# in that case save it as an image
jpeg('example2-network.jpeg')
network(nutri.res, comp = 1:3, cutoff = 0.45,
color.node = c("mistyrose", "lightcyan"),
shape.node = c("circle", "rectangle"),
color.edge = color.jet(100),
lty.edge = "solid", lwd.edge = 2,
show.edge.labels = FALSE)
dev.off()

## End(Not run)

## interactive 'cutoff'
## Not run:
network(nutri.res, comp = 1:3, cutoff = 0.55, interactive = TRUE)
## select the 'cutoff' and "see" the new network

## End(Not run)

## network representation for objects of class 'spls'
data(liver.toxicity)
X <- liver.toxicity$gene
Y <- liver.toxicity$clinic
toxicity.spls <- spls(X, Y, ncomp = 3, keepX = c(50, 50, 50),
keepY = c(10, 10, 10))

## Not run:
# sometimes with Rstudio might not work because of margin issues,
# in that case save it as an image
jpeg('example3-network.jpeg')
network(toxicity.spls, comp = 1:3, cutoff = 0.8,
color.node = c("mistyrose", "lightcyan"),
shape.node = c("circle", "rectangle"),
color.edge = color.jet(100),
lty.edge = "solid", lwd.edge = 2,
show.edge.labels = FALSE)
dev.off()
nipals  

Non-linear Iterative Partial Least Squares (NIPALS) algorithm

Description

This function performs NIPALS algorithm, i.e. the singular-value decomposition (SVD) of a data table that can contain missing values.

Usage

nipals(X, ncomp = 1, reconst = FALSE, max.iter = 500, tol = 1e-09)

Arguments

X  real matrix or data frame whose SVD decomposition is to be computed. It can contain missing values.

ncomp  integer, the number of components to keep. If missing ncomp=ncol(X).

reconst  logical that specify if nipals must perform the reconstitution of the data using the ncomp components.

max.iter  integer, the maximum number of iterations.

tol  a positive real, the tolerance used in the iterative algorithm.

Details

The NIPALS algorithm (Non-linear Iterative Partial Least Squares) has been developed by H. Wold at first for PCA and later-on for PLS. It is the most commonly used method for calculating the principal components of a data set. It gives more numerically accurate results when compared with the SVD of the covariance matrix, but is slower to calculate.

This algorithm allows to realize SVD with missing data, without having to delete the rows with missing data or to estimate the missing data.

Value

The returned value is a list with components:

eig  vector containing the pseudosingular values of X, of length ncomp.

t  matrix whose columns contain the left singular vectors of X.

p  matrix whose columns contain the right singular vectors of X.

rec  matrix obtained by the reconstitution of the data using the ncomp components.
Author(s)
Sebastien Dejean and Ignacio Gonzalez.

References

See Also
svd, princomp, prcomp, eigen and http://www.mixOmics.org for more details.

Examples

```r
## Hilbert matrix
hilbert <- function(n) { i <- 1:n; 1 / outer(i - 1, i, "+") }
X.na <- X <- hilbert(9)[, 1:6]

## Hilbert matrix with missing data
idx.na <- matrix(sample(c(0, 1, 1, 1, 1, 1), 36, replace = TRUE), ncol = 6)
X.na[idx.na == 0] <- NA
X.rec <- nipals(X.na, reconst = TRUE)$rec
round(X, 2)
round(X.rec, 2)
```

Description
The `nutrimouse` dataset contains the expression measure of 120 genes potentially involved in nutritional problems and the concentrations of 21 hepatic fatty acids for forty mice.

Usage
data(nutrimouse)

Format
A list containing the following components:
gene data frame with 40 observations on 120 numerical variables.
lipid data frame with 40 observations on 21 numerical variables.
diet factor of 5 levels containing 40 labels for the diet factor.
genotype factor of 2 levels containing 40 labels for the diet factor.
Details

The data sets come from a nutrigenomic study in the mouse (Martin et al., 2007) in which the effects of five regimens with contrasted fatty acid compositions on liver lipids and hepatic gene expression in mice were considered. Two sets of variables were acquired on forty mice:

- gene: expressions of 120 genes measured in liver cells, selected (among about 30,000) as potentially relevant in the context of the nutrition study. These expressions come from a nylon macroarray with radioactive labelling;
- lipid: concentrations (in percentages) of 21 hepatic fatty acids measured by gas chromatography.

Biological units (mice) were cross-classified according to two factors experimental design (4 replicates):

- Genotype: 2-levels factor, wild-type (WT) and PPARα -/- (PPAR).
- Diet: 5-levels factor. Oils used for experimental diets preparation were corn and colza oils (50/50) for a reference diet (REF), hydrogenated coconut oil for a saturated fatty acid diet (COC), sunflower oil for an Omega6 fatty acid-rich diet (SUN), linseed oil for an Omega3-rich diet (LIN) and corn/colza/enriched fish oils for the FISH diet (43/43/14).

Source

The nutrimouse dataset was provided by Pascal Martin from the Toxicology and Pharmacology Laboratory, National Institute for Agronomic Research, French.

References


pca

Principal Components Analysis

Description

Performs a principal components analysis on the given data matrix that can contain missing values. If data are complete 'pca' uses Singular Value Decomposition, if there are some missing values, it uses the NIPALS algorithm.
Usage

```r
pca(X,
   ncomp = 2,
   center = TRUE,
   scale = FALSE,
   max.iter = 500,
   tol = 1e-09,
   logratio = 'none', # one of ('none','CLR','ILR')
   ilr.offset = 0.001,
   V = NULL,
   multilevel = NULL)
```

Arguments

- **X**  
a numeric matrix (or data frame) which provides the data for the principal components analysis. It can contain missing values.

- **ncomp**  
integer, if data is complete ncomp decides the number of components and associated eigenvalues to display from the pcasvd algorithm and if the data has missing values, ncomp gives the number of components to keep to perform the reconstitution of the data using the NIPALS algorithm. If NULL, function sets ncomp = min(nrow(X), ncol(X))

- **center**  
a logical value indicating whether the variables should be shifted to be zero centered. Alternately, a vector of length equal the number of columns of X can be supplied. The value is passed to scale.

- **scale**  
a logical value indicating whether the variables should be scaled to have unit variance before the analysis takes place. The default is FALSE for consistency with princomp function, but in general scaling is advisable. Alternatively, a vector of length equal the number of columns of X can be supplied. The value is passed to scale.

- **max.iter**  
integer, the maximum number of iterations in the NIPALS algorithm.

- **tol**  
a positive real, the tolerance used in the NIPALS algorithm.

- **logratio**  
one of ('none','CLR','ILR'). Specifies the log ratio transformation to deal with compositional values that may arise from specific normalisation in sequencing data. Default to 'none'

- **ilr.offset**  
When logratio is set to 'ILR', an offset must be input to avoid infinite value after the logratio transform, default to 0.001.

- **V**  
Matrix used in the logratio transformation id provided.

- **multilevel**  
sample information for multilevel decomposition for repeated measurements.

Details

The calculation is done either by a singular value decomposition of the (possibly centered and scaled) data matrix, if the data is complete or by using the NIPALS algorithm if there is data missing. Unlike princomp, the print method for these objects prints the results in a nice format and the plot method produces a bar plot of the percentage of variance explained by the principal components (PCs).
When using NIPALS (missing values), we make the assumption that the first \( \min(\text{ncol}(X), \text{nrow}(X)) \) principal components will account for 100% of the explained variance. Note that \text{scale} = \text{TRUE} cannot be used if there are zero or constant (for \text{center} = \text{TRUE}) variables. Components are omitted if their standard deviations are less than or equal to \text{comp.tol} times the standard deviation of the first component. With the default null setting, no components are omitted. Other settings for \text{comp.tol} could be \text{comp.tol} = \text{sqrt(Machine\$double.eps)}, which would omit essentially constant components, or \text{comp.tol} = 0.

According to Filzmoser et al., a ILR log ratio transformation is more appropriate for PCA with compositional data. Both CLR and ILR are valid.

Logratio transform and multilevel analysis are performed sequentially as internal pre-processing step, through \text{logratio.transfo} and \text{withinVariation} respectively.

Logratio can only be applied if the data do not contain any 0 value (for count data, we thus advise normalise raw data with a 1 offset). For ILR transformation and additional offset might be needed.

\textbf{Value}

\texttt{pca} returns a list with class "\texttt{pca}" and "\texttt{prcomp}" containing the following components:

\begin{itemize}
\item \texttt{ncomp} the number of principal components used.
\item \texttt{sdev} the eigenvalues of the covariance/correlation matrix, though the calculation is actually done with the singular values of the data matrix or by using NIPALS.
\item \texttt{rotation} the matrix of variable loadings (i.e., a matrix whose columns contain the eigenvectors).
\item \texttt{loadings} same as 'rotation' to keep the mixOmics spirit
\item \texttt{x} the value of the rotated data (the centred (and scaled if requested) data multiplied by the rotation/loadings matrix), also called the principal components.
\item \texttt{variates} same as 'x' to keep the mixOmics spirit
\item \texttt{center}, \texttt{scale} the centering and scaling used, or \texttt{FALSE}.
\item \texttt{explained_variance} explained variance from the multivariate model, used for \texttt{plotIndiv}
\end{itemize}

\textbf{Author(s)}

Florian Rohart, Kim-Anh Le Cao, Ignacio Gonzalez

\textbf{References}

pcatune

Tune the number of principal components in PCA

Description

This function has been renamed tune.pca.

See Also

nipals, prcomp, biplot, plotIndiv, plotVar and http://www.mixOmics.org for more details.

Examples

```r
# example with missing values where NIPALS is applied
# -----------------------------------------------
data(multidrug)
pca.res <- pca(multidrug$ABC.trans, ncomp = 4, scale = TRUE)
plot(pca.res)
print(pca.res)
biplot(pca.res, xlabs = multidrug$cell.line$Class, cex = 0.7)

# samples representation
plotIndiv(pca.res, ind.names = multidrug$cell.line$Class,
           group = as.numeric(as.factor(multidrug$cell.line$Class)))
## Not run:
plotIndiv(pca.res, cex = 0.2,
          col = as.numeric(as.factor(multidrug$cell.line$Class)),style="3d")

## End(Not run)
# variable representation
plotVar(pca.res)
## Not run:
plotVar(pca.res, rad.in = 0.5, cex = 0.5,style="3d")

## End(Not run)

# example with multilevel decomposition and CLR log ratio transformation (ILR longer to run)
# -------------------------------
## Not run:
data("diverse.16S")
pca.res = pca(X = diverse.16S$data.TSS, ncomp = 5,
               logratio = 'CLR', multilevel = diverse.16S$data)
plot(pca.res)
plotIndiv(pca.res, ind.names = FALSE, group = diverse.16S$data$sample, title = '16S diverse data',
          legend = TRUE)

## End(Not run)
```
Description

Function to evaluate the performance of the fitted PLS, sparse PLS, PLS-DA, sparse PLS-DA, MINT (mint.splsda) and DIABLO (block.splsda) models using various criteria.

Usage

## S3 method for class 'pls'
perf(object, validation = c("Mfold", "loo"),
     folds = 10, progressBar = TRUE, ...)

## S3 method for class 'spls'
perf(object, validation = c("Mfold", "loo"),
     folds = 10, progressBar = TRUE, ...)

## S3 method for class 'plsda'
perf(object,
     dist = c("all", "max.dist", "centroids.dist", "mahalanobis.dist"),
     validation = c("Mfold", "loo"),
     folds = 10, nrepeat =1, auc = FALSE, progressBar = TRUE, cpus, ...)

## S3 method for class 'splsda'
perf(object,
     dist = c("all", "max.dist", "centroids.dist", "mahalanobis.dist"),
     validation = c("Mfold", "loo"),
     folds = 10, nrepeat =1, auc = FALSE, progressBar = TRUE, cpus, ...)

## S3 method for class 'mint.splsda'
perf(object,
     dist = c("all", "max.dist", "centroids.dist", "mahalanobis.dist"),
     auc = FALSE, progressBar = TRUE, ...)

## S3 method for class 'sgccda'
perf(object,
     dist = c("all", "max.dist", "centroids.dist", "mahalanobis.dist"),
     validation = c("Mfold", "loo"),
     folds = 10, nrepeat =1, cpus, ...)

Arguments

object object of class inheriting from "pls", "plsda", "spls", "splsda" or "mint.splsda". The function will retrieve some key parameters stored in that object.
perf

99

dist

only applies to an object inheriting from "plsda", "splsda" or "mint.splsda" to evaluate the classification performance of the model. Should be a subset of "max.dist", "centroids.dist", "mahalanobis.dist". Default is "all." See predict.

validation

character. What kind of (internal) validation to use, matching one of "Mfold" or "loo" (see below). Default is "Mfold".

folds

the folds in the Mfold cross-validation. See Details.

nrepeat

Number of times the Cross-Validation process is repeated. This is an important argument to ensure the estimation of the performance to be as accurate as possible.

auc

if TRUE calculate the Area Under the Curve (AUC) performance of the model.

progressBar

by default set to TRUE to output the progress bar of the computation.

cpus

Number of cpus to use when running the code in parallel.

... not used

Details

Procedure. The process of evaluating the performance of a fitted model object is similar for all PLS-derived methods; a cross-validation approach is used to fit the method of object on folds-1 subsets of the data and then to predict on the subset left out. Different measures of performance are available depending on the model. Parameters such as logratio, multilevel, keepX or keepY are retrieved from object.

Parameters. If validation = "Mfold", M-fold cross-validation is performed. folds specifies the number of folds to generate. The folds also can be supplied as a list of vectors containing the indexes defining each fold as produced by split. When using validation = "Mfold", make sure that you repeat the process several times (as the results will be highly dependent on the random splits and the sample size).

If validation = "loo", leave-one-out cross-validation is performed (in that case, there is no need to repeat the process).

Measures of performance. For fitted PLS and sPLS regression models, perf estimates the mean squared error of prediction (MSEP), $R^2$, and $Q^2$ to assess the predictive perfority of the model using M-fold or leave-one-out cross-validation. Note that only the classic, regression and invariant modes can be applied. For sPLS, the MSEP, $R^2$, and $Q^2$ criteria are averaged across all folds. Note that for PLS and sPLS objects, perf is performed on the pre-processed data after log ratio transform and multilevel analysis, if any.

Sparse methods. The sPLS, sPLS-DA and sgccda functions are run on several and different subsets of data (the cross-folds) and will certainly lead to different subset of selected features. Those are summarised in the output featuresDstable (see output Value below) to assess how often the variables are selected across all folds. Note that for PLS-DA and sPLS-DA objects, perf is performed on the original data, i.e. before the pre-processing step of the log ratio transform and multilevel analysis, if any. In addition for these methods, the classification error rate is averaged across all folds.

The mint.sPLS-DA function estimates errors based on Leave-one-group-out cross validation (where each levels of object$study is left out (and predicted) once) and provides study-specific outputs (study-specific.error) as well as global outputs (global.error).
AUROC. For PLS-DA, sPLS-DA, mint.PLS-DA and mint.sPLS-DA methods: if auc=TRUE, Area Under the Curve (AUC) values are calculated from the predicted scores obtained from the predict function applied to the internal test sets in the cross-validation process, either for all samples or for study-specific samples (for mint models). Therefore we minimise the risk of overfitting. See auroc for more details. Our multivariate supervised methods already use a prediction threshold based on distances (see predict) that optimally determine class membership of the samples tested. As such AUC and ROC are not needed to estimate the performance of the model. We provide those outputs as complementary performance measures. See more details in our mixOmics article.

Prediction distances. See details from ?predict.

Repeats of the CV-folds. Repeated cross-validation implies that the whole CV process is repeated a number of times (nrepeat) to reduce variability across the different subset partitions. In the case of Leave-One-Out CV (validation = 'loo'), each sample is left out once (folds = N is set internally) and therefore nrepeat is by default 1.

More details about the PLS modes in ?pls.

Value

For PLS and sPLS models, perf produces a list with the following components:

- **MSEP**  
  Mean Square Error Prediction for each Y variable, only applies to object inherited from "pls", and "spls".

- **R2**  
  A matrix of $R^2$ values of the Y-variables for models with 1,...,ncomp components, only applies to object inherited from "pls", and "spls".

- **Q2**  
  If Y contains one variable, a vector of $Q^2$ values else a list with a matrix of $Q^2$ values for each Y-variable. Note that in the specific case of an sPLS model, it is better to have a look at the Q2.total criterion, only applies to object inherited from "pls", and "spls"

- **Q2.total**  
  A vector of $Q^2$-total values for models with 1,...,ncomp components, only applies to object inherited from "pls", and "spls"

- **features**  
  A list of features selected across the folds ($stable.X$ and $stable.Y$) for the keepx and keepy parameters from the input object.

- **error.rate**  
  For PLS-DA and sPLS-DA models, perf produces a matrix of classification error rate estimation. The dimensions correspond to the components in the model and to the prediction method used, respectively. Note that error rates reported in any component include the performance of the model in earlier components for the specified keepx parameters (e.g. error rate reported for component 3 for keepx = 20 already includes the fitted model on components 1 and 2 for keepx = 20). For more advanced usage of the perf function, see www.mixomics.org/methods/spls-da/ and consider using the predict function.

- **auc**  
  Averaged AUC values over the nrepeat

For mint.splsda models, perf produces the following outputs:

- **study_specific.error**  
  A list that gives BER, overall error rate and error rate per class, for each study

- **global.error**  
  A list that gives BER, overall error rate and error rate per class for all samples
predict  A list of length ncomp that produces the predicted values of each sample for each class
class    A list which gives the predicted class of each sample for each dist and each of the ncomp components. Directly obtained from the predict output.
auc      AUC values
auc.study AUC values for each study

For sgccda models, perf produces the following outputs:

error.rate Prediction error rate for each block of object$X$ and each dist
error.rate.per.class Prediction error rate for each block of object$X$, each dist and each class
predict    Predicted values of each sample for each class, each block and each component
class      Predicted class of each sample for each block, each dist, each component and each nrepeat
features    a list of features selected across the folds ($stable.X$ and $stable.Y$) for the keepX and keepY parameters from the input object.

AveragedPredict.class if more than one block, returns the average predicted class over the blocks (averaged of the Predict output and prediction using the max.dist distance)
AveragedPredict.error.rate if more than one block, returns the average predicted error rate over the blocks (using the AveragedPredict.class output)

WeightedPredict.class if more than one block, returns the weighted predicted class over the blocks (weighted average of the Predict output and prediction using the max.dist distance)
WeightedPredict.error.rate if more than one block, returns the weighted average predicted error rate over the blocks (using the WeightedPredict.class output)

MajorityVote    if more than one block, returns the majority class over the blocks. NA for a sample means that there is no consensus on the predicted class for this particular sample over the blocks.
MajorityVote.error.rate if more than one block, returns the error rate of the MajorityVote output

WeightedVote    if more than one block, returns the weighted majority class over the blocks. NA for a sample means that there is no consensus on the predicted class for this particular sample over the blocks.
WeightedVote.error.rate if more than one block, returns the error rate of the WeightedVote output

weights    Returns the weights of each block used for the weighted predictions, for each nrepeat and each fold

choice.ncomp For supervised models; returns the optimal number of components for the model for each prediction distance using one-sided t-tests that test for a significant difference in the mean error rate (gain in prediction) when components are added to the model. See more details in Rohart et al 2016 Suppl. For more than one block, an optimal ncomp is returned for each prediction framework.
Author(s)


References

DIABLO:

mixOmics main publication:

MINT:


Chavent, Marie and Patouille, Brigitte (2003). Calcul des coefficients de regression et du PRESS en regression PLS1. *Modulad n*, **30** 1-11. (this is the formula we use to calculate the Q2 in perf.pls and perf.spls)


sparse PLS regression mode:

One-sided t-tests (suppl material):

See Also

`predict`, `nipals`, `plot.perf`, `auroc` and [www.mixOmics.org](http://www.mixOmics.org) for more details.

Examples

```r
## Not run:
## validation for objects of class 'pls' (regression)
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
data(liver.toxicity)
X <- liver.toxicity$gene
Y <- liver.toxicity$clinic
```
# try tune the number of component to choose
# ---------------------------
# first learn the full model
liver.pls <- pls(X, Y, ncomp = 10)

# with 5-fold cross validation: we use the same parameters as in model above
# but we perform cross validation to compute the MSE, Q2 and R2 criteria
# ---------------------------
liver.val <- perf(liver.pls, validation = "Mfold", folds = 5)

# Q2 total should decrease until it reaches a threshold
liver.val$Q2.total

# ncomp = 2 is enough
plot(liver.val$Q2.total, type = 'l', col = 'red', ylim = c(0.5, 0.5),
     xlab = 'PLS components', ylab = 'Q2 total')
abline(h = 0.0975, col = 'darkgreen')
legend('topright', col = c('red', 'darkgreen'),
       legend = c('Q2 total', 'threshold 0.0975'), lty = 1)
title('Liver toxicity PLS 5-fold, Q2 total values')

# have a look at the other criteria
# ---------------------------
# R2
liver.val$R2

matplot(t(liver.val$R2), type = 'l', xlab = 'PLS components', ylab = 'R2 for each variable')
title('Liver toxicity PLS 5-fold, R2 values')

# MSE
liver.val$MSEP

matplot(t(liver.val$MSEP), type = 'l', xlab = 'PLS components', ylab = 'MSEP for each variable')
title('Liver toxicity PLS 5-fold, MSE values')

## validation for objects of class 'spls' (regression)
## ---------------------------
ncomp = 7
# first, learn the model on the whole data set
model.spls = spls(X, Y, ncomp = ncomp, mode = 'regression',
                   keepX = c(rep(10, ncomp)), keepY = c(rep(4, ncomp)))

# with leave-one-out cross validation
#set.seed(45)
model.spls.val <- perf(model.spls, validation = "Mfold", folds = 5 )#validation = "loo")

# Q2 total
model.spls.val$Q2.total

# R2: we can see how the performance degrades when ncomp increases
model.spls.val$R2
plot(model.spls.val, criterion="R2", type = 'l')
plot(model.spls.val, criterion="Q2", type = '1')

## validation for objects of class 'splsda' (classification)
## -----------------------------------------------
data(srbct)
X <- srbct$gene
Y <- srbct$class

ncomp = 2
srbct.splsda <- splsda(X, Y, ncomp = ncomp, keepX = rep(10, ncomp))

# with Mfold
# ------
set.seed(45)
error <- perf(srbct.splsda, validation = "Mfold", folds = 8,
dist = "all", auc = TRUE)
error
error$auc

plot(error)

# parallel code
set.seed(45)
error <- perf(srbct.splsda, validation = "Mfold", folds = 8,
dist = "all", auc = TRUE, cpus =2)

\dontrun{
  # with 5 components and nrepeat =5, to get a choice.ncomp
  ncomp = 5
  srbct.splsda <- splsda(X, Y, ncomp = ncomp, keepX = rep(10, ncomp))

  set.seed(45)
  error <- perf(srbct.splsda, validation = "Mfold", folds = 8,
dist = "all", nrepeat =5)
  error

  plot(error)

  # parallel code
  set.seed(45)
  error <- perf(srbct.splsda, validation = "Mfold", folds = 8,
dist = "all", auc = TRUE, cpus =2)
}

## validation for objects of class 'mint.splsda' (classification)
## -----------------------------------------------
```r
data(stemcells)
res = mint.splsda(X = stemcells$gene, Y = stemcells$celltype, ncomp = 3, keepX = c(10, 5, 15),
study = stemcells$study)

out = perf(res, auc = TRUE)
out
out$auc
out$auc.study

## validation for objects of class 'sgccda' (classification)
# ------------------------------------------

data(nutrimouse)
Y = nutrimouse$diet
data = list(gene = nutrimouse$gene, lipid = nutrimouse$lipid)
design = matrix(c(0,1,1,1,0,1,1,0), ncol = 3, nrow = 3, byrow = TRUE)
nutrimouse.sgccda <- block.splsda(X=data,
Y = Y,
design = design,
keepX = list(gene=c(10,10), lipid=c(15,15)),
ncomp = 2,
scheme = "horst",
verbose = FALSE,
bias = FALSE)

perf = perf(nutrimouse.sgccda)
perf

\dontrun{
#with 5 components and nrepeat=5 to get $choice.ncomp
nutrimouse.sgccda <- block.splsda(X=data,
Y = Y,
design = design,
keepX = list(gene=c(10,10), lipid=c(15,15)),
ncomp = 5,
scheme = "horst",
verbose = FALSE,
bias = FALSE)

perf = perf(nutrimouse.sgccda, folds = 5, nrepeat = 5)
perf

perf$choice.ncomp
}

## End(Not run)
```
Description

Function to plot performance criteria, such as MSEP, RMSEP, $R^2$, $Q^2$ for s/PLS methods, and classification performance for supervised methods, as a function of the number of components.

Usage

```r
# S3 method for class 'perf.spls.mthd'
plot(x,
criterion = "MSEP",
xlab = "number of components",
ylab = NULL,
LimQ2 = 0.0975,
LimQ2.col = "darkgrey",
cTicks = NULL,
layout = NULL,
...)

# S3 method for class 'perf.splsda.mthd'
plot(x,
dist = c("all","max.dist","centroids.dist","mahalanobis.dist"),
measure = c("all","overall","BER"),
col,
xlab = NULL,
ylab = NULL,
overlay = c("all", "measure", "dist"),
legend.position = c("vertical", "horizontal"),
sd = TRUE,
...)

# S3 method for class 'perf.mint.splsda.mthd'
plot(x,
dist = c("all","max.dist","centroids.dist","mahalanobis.dist"),
measure = c("all","overall","BER"),
col,
xlab = NULL,
ylab = NULL,
study = "global",
overlay = c("all", "measure", "dist"),
legend.position = c("vertical", "horizontal"),
...)

# S3 method for class 'perf.sgclda.mthd'
plot(x,
```
Arguments

x an perf object.
criterion character string. What type of validation criterion to plot for pls or spls. One of "MSEP", "RMSEP", "R2" or "Q2". See perf.
dist prediction method applied in perf for plsda or splsda. See perf.
measure Two misclassification measure are available: overall misclassification error overall or the Balanced Error Rate BER
col character (or symbol) color to be used, possibly vector. One color per distance dist.
weighted plot either the performance of the Majority vote or the Weighted vote.
study Indicates which study-specific outputs to plot. A character vector containing some levels of object$study, "all.partial" to plot all studies or "global" is expected. Default to "global".
overlay parameter to overlay graphs; if 'all', only one graph is shown with all outputs; if 'measure', a graph is shown per distance; if 'dist', a graph is shown per measure.
legend.position position of the legend, one of "vertical" (only one column) or "horizontal" (two columns).
xlab, ylab titles for x and y axes. Typically character strings, but can be expressions (e.g., expression(R^2)).
LimQ2 numeric value. Signification limit for the components in the model. Default is LimQ2 = 0.0975.
LimQ2.col character string specifying the color for the LimQ2 line to be plotted. If "none" the line will not be plotted.
cTicks integer vector. Axis tickmark locations for the used number of components. Default is 1:ncomp (see perf).
layout numeric vector of length two giving the number of rows and columns in a multi panel display. If not specified, plot.perf tries to be intelligent.
sd If 'nrepeat' was used in the call to 'perf', error bar shows the standard deviation if sd=TRUE
... Further arguments sent to xyplot function.
Details

plot.perf creates one plot for each response variable in the model, laid out in a multi panel display. It uses xyplot for performing the actual plotting.

More details about the prediction distances in ?predict and the referred publication.

Author(s)


References


See Also

pls, spls, plsda, splsda, perf.

Examples

require(lattice)

## validation for objects of class 'pls' or 'spls'
## Not run:
data(liver.toxicity)
X <- liver.toxicity$gene
Y <- liver.toxicity$clinic

liver.pls <- pls(X, Y, ncomp = 3)
liver.perf <- perf(liver.pls, validation = "Mfold")

plot(liver.perf, criterion = "R2", layout = c(2, 2))

## End(Not run)

## validation for objects of class 'plsda' or 'splsda'
## Not run:
data(breast.tumors)
X <- breast.tumors$gene.exp
# Y will be transformed as a factor in the function,
# but we set it as a factor to set up the colors.
Y <- as.factor(breast.tumors$sample$treatment)

res <- splsda(X, Y, ncomp = 2, keepX = c(25, 25))
breast.perf <- perf(res, nrepeat = 5)

plot(breast.perf)
plot(breast.perf, col=1:3)
plot(breast.perf, col=1:3, sd=FALSE)
Description

This function provides scree plot of the canonical correlations.

Usage

## S3 method for class 'rcc'
plot(x, scree.type = c("pointplot", "barplot"), ...)

Arguments

x          object of class inheriting from "rcc".
scree.type character string, (partially) matching one of "pointplot" or "barplot", determining the kind of scree plots to be produced.
...        arguments to be passed to other methods. For the "pointplot" type see `points`, for "barplot" type see `barplot`.

Author(s)

Sebastien Dejean and Ignacio Gonzalez.

See Also

`points`, `barplot`, `par`.

Examples

data(nutrimouse)
X <- nutrimouse$lipid
Y <- nutrimouse$gene
nutri.res <- rcc(X, Y, lambda1 = 0.064, lambda2 = 0.008)

## 'pointplot' type scree
plot(nutri.res) #(default)

plot(nutri.res, pch = 19, cex = 1.2,
     col = c(rep("red", 3), rep("darkblue", 18)))

## 'barplot' type scree
plot(nutri.res, scree.type = "barplot")

plot(nutri.res, scree.type = "barplot", density = 20, col = "black")
Description

Function to plot performance criteria, such as classification error rate or balanced error rate on a tune.splsda result.

Usage

```r
## S3 method for class 'tune.splsda'
plot(x, optimal = TRUE, sd = TRUE, legend.position = "topright", col, ...)
```

Arguments

- `x`: an tune.splsda object.
- `optimal`: If TRUE, highlights the optimal keepX per component
- `sd`: If 'nrepeat' was used in the call to 'tune.splsda', error bar shows the standard deviation if sd=TRUE
- `legend.position`: position of the legend, one of "bottomright", "bottom", "bottomleft", "left", "topleft", "top", "topright", "right" and "center".
- `col`: character (or symbol) color to be used, possibly vector. One color per component.
- `...`: Further arguments sent to `xyplot` function.

Details

`plot.tune.splsda` plots the classification error rate or the balanced error rate from x$error.rate, for each component of the model. A circle highlights the optimal number of variables on each component.

Author(s)

Kim-Anh Le Cao, Florian Rohart, Francois Bartolo.

See Also

Examples

```r
## validation for objects of class 'splsda'
## Not run:
data(breast.tumors)
X = breast.tumors$gene.exp
Y = as.factor(breast.tumors$sample$treatment)
out = tune.splsda(X, Y, ncomp = 3, nrepeat = 2, logratio = "none",
test.keepX = c(5, 10, 15), folds = 10, dist = "max.dist",
progressBar = TRUE)

plot(out)

## End(Not run)

## validation for objects of class 'mint.splsda'
## Not run:
data(stemcells)
data = stemcells$gene
type.id = stemcells$celltype
exp = stemcells$study

out = tune(method="mint.splsda", X=data,Y=type.id, ncomp=2, study=exp, test.keepX=seq(1,10,1))
out$choice.keepX.constraint

plot(out)

## End(Not run)
```

plotArrow

---

**Description**

Represents samples from multiple coordinates.

**Usage**

```
plotArrow(  object,
        comp = NULL,
        abline = FALSE,
        xlim = NULL,
        ylim = NULL,
        group=NULL,
        col,
        cex,
```

---

Arrow sample plot
plotArrow

pch, title=NULL, plot.arrows=TRUE, legend=FALSE, X.label = NULL, Y.label = NULL, ind.names=FALSE, position.names='centroid'

Arguments

object object of class inheriting from mixOmics: PLS, sPLS, rCC, rGCCA, sGCCA, sGCCDA,
comp integer vector of length two indicating the components represented on the horizontal and the vertical axis to project the individuals.
abline should the vertical and horizontal line through the center be plotted? Default set to FALSE
xlim the ranges to be encompassed by the x axis, if NULL they are computed.
ylim the ranges to be encompassed by the y axis, if NULL they are computed.
group factor indicating the group membership for each sample. Coded as default for the supervised method sGCCDA, sPLSDA, but needs to be input for the unsupervised methods PLS, sPLS, rCC, rGCCA, sGCCA
col character (or symbol) color to be used, color vector also possible.
cex numeric character (or symbol) expansion, , color vector also possible.
pch plot character. A character string or a vector of single characters or integers. See points for all alternatives.
title set of characters for the title plot.
plot.arrows boolean. Whether arrows should be added or not. Default is TRUE.
legend boolean. Whether the legend should be added. Only for the supervised methods and if group!=NULL. Default is FALSE.
X.label x axis titles.
Y.label y axis titles.
ind.names If TRUE, the row names of the first (or second) data matrix are used as sample names (see Details). Can be a vector of length the sample size to display sample names.
position.names One of "centroid", "start", "end". Define where sample names are plotted when ind.names=TRUE. In a multiblock analysis, centroid and start will display similarly.

Details

Graphical of the samples (individuals) is displayed in a superimposed manner where each sample will be indicated using an arrow. The start of the arrow indicates the location of the sample in X in one plot, and the tip the location of the sample in Y in the other plot.
For objects of class "GCCA" and if there are more than 3 blocks, the start of the arrow indicates the centroid between all data sets for a given individual and the tips of the arrows the location of that individual in each block.

Short arrows indicate a strong agreement between the matching data sets, long arrows a disagreement between the matching data sets.

Author(s)

Francois Bartolo, Kim-Anh Le Cao.

References


See Also

`arrows`, `text`, `points` and http://mixOmics.org/graphics for more details.

Examples

```r
## plot of individuals for objects of class 'rcc'
# -----------------------------------------------
data(nutrimouse)
X <- nutrimouse$lipid
Y <- nutrimouse$gene
nutri.res <- rcc(X, Y, ncomp = 3, lambda1 = 0.064, lambda2 = 0.008)
plotArrow(nutri.res)

# names indicate genotype
plotArrow(nutri.res,
  group = nutrimouse$genotype, ind.names = nutrimouse$genotype)

## Not run:
plotArrow(nutri.res, group = nutrimouse$genotype,
  legend = TRUE)

## End(Not run)

## plot of individuals for objects of class 'pls' or 'spls'
# -----------------------------------------------
data(liver.toxicity)
X <- liver.toxicity$gene
Y <- liver.toxicity$clinic
toxicity.spls <- spls(X, Y, ncomp = 3, keepX = c(50, 50, 50),
  keepY = c(10, 10, 10))

#default
plotArrow(toxicity.spls)
```
## Not run:

`plotArrow(toxicity.spls, group = liver.toxicity$treatment[, 'Time.Group'],
ind.names = liver.toxicity$treatment[, 'Dose.Group'],
legend = TRUE)`

```
# colors indicate time of necropsy, text is the dose
plotArrow(toxicity.spls, group = liver.toxicity$treatment[, 'Time.Group'],
ind.names = liver.toxicity$treatment[, 'Dose.Group'],
legend = TRUE, position.names = 'start')
```

## End(Not run)

```
## variable representation for objects of class 'sgcca' or 'rgcca'
## -----------------------------------------------
data(nutrimouse)
Y = unmap(nutrimouse$diet)
data = list(gene = nutrimouse$gene, lipid = nutrimouse$lipid, Y = Y)
design1 = matrix(c(0,1,1,1,0,1,1,1,0), ncol = 3, nrow = 3, byrow = TRUE)
nutrimouse.sgcca <- wrapper.sgcca(X = data,
design = design1,
penalty = c(0.3, 0.5, 1),
ncomp = 3,
scheme = "centroid",
verbose = FALSE,
bias = FALSE)
```

## Not run: same color for all samples

```
plotArrow(nutrimouse.sgcca)
```

```
# ind.names to visualise the unique individuals
plotArrow(nutrimouse.sgcca, group = nutrimouse$diet, legend =TRUE,
title = 'my plot', ind.names = TRUE)
```

```
# ind.names to visualise the unique individuals
plotArrow(nutrimouse.sgcca, group = nutrimouse$diet, legend =TRUE,
title = 'my plot', ind.names = TRUE, position.names = 'start')
```

```
plotArrow(nutrimouse.sgcca, group = nutrimouse$diet, legend =TRUE,
title = 'my plot', ind.names = TRUE, position.names = 'end')
```

```
# ind.names indicates the diet
plotArrow(nutrimouse.sgcca, group = nutrimouse$diet, legend =TRUE,
title = 'my plot', ind.names = nutrimouse$diet, position.names = 'start')
```

```
# ind.names to visualise the unique individuals, start position
plotArrow(nutrimouse.sgcca, group = nutrimouse$diet, legend =TRUE,
title = 'my plot', ind.names = TRUE, position.names = 'start')
```

plotContribution

# variable representation for objects of class 'sgccda'
# Note: the code differs from above as we use a 'supervised' GCCA analysis

data(nutrimouse)
Y = nutrimouse$diet
data = list(gene = nutrimouse$gene, lipid = nutrimouse$lipid)
design1 = matrix(c(0,1,0,1), ncol = 2, nrow = 2, byrow = TRUE)

nutrimouse.sgccda1 <- wrapper.sgccda(X = data,
Y = Y,
design = design1,
ncomp = 2,
keepX = list(gene = c(10,10), lipid = c(15,15)),
scheme = "centroid",
verbose = FALSE,
bias = FALSE)

# default colors correspond to outcome Y
plotArrow(nutrimouse.sgccda1)

## Not run:
## with legend and title and indiv ID
plotArrow(nutrimouse.sgccda1, legend = TRUE, title = 'my sample plot',
ind.names = TRUE, position.names = 'start')

## End(Not run)

plotContribution

Description

This function provides a horizontal bar plot to visualise the highest or lowest mean/median value of the variables with color code corresponding to the outcome of interest. Applies DA objects (sPLS-DA, PLS-DA, and multilevel, SGCCA) and for count data types.

Usage

plotContribution(object,
    contrib="max", # choose between 'max' or 'min'
    method = "mean", # choose between 'mean' or 'median'
    block=1, # single value
    comp = 1,
plot = TRUE,
show.ties = TRUE,
col.ties="white",
ndisplay = NULL,
cex.name = 0.7,
cex.legend = 0.8,
name.var=NULL,
name.var.complete=FALSE,
title = NULL,
legend = TRUE,
legend.color = NULL,
legend.title = 'Outcome'
}

Arguments

object  
object of class inheriting from a plsda, splsda or sgccda class.

contrib  
a character set to 'max' or 'min' indicating if the color of the bar should correspond to the group with the maximal or minimal expression levels / abundance. By default set to 'max'.

method  
a character set to 'mean' or 'median' indicating the criterion to assess the contribution. We recommend using median in the case of count or skewed data.

block  
a single value indicating which block to consider in a sgccda object. Default is set to the first block.

comp  
integer value indicating the component of interest from the object.

plot  
Boolean indicating of the plot should be output. If set to FALSE the user can extract the contribution matrix, see example. Default value is TRUE.

show.ties  
Boolean. If TRUE then tie groups appear in the color set by col.ties, which will appear in the legend. Ties can happen when dealing with count data type. By default set to TRUE.

col.ties  
Color corresponding to ties, only used if show.ties=TRUE and ties are present.

ndisplay  
integer indicating how many of the most important variables are to be plotted (ranked by decreasing weights in each PLS-component). Useful to lighten a graph.

cex.name  
a numerical value giving the amount by which plotting the variable name text should be magnified or reduced relative to the default.

cex.legend  
a numerical value giving the amount by which plotting the legend text should be magnified or reduced relative to the default.

name.var  
a character vector indicating the names of the variables. The names of the vector should match the names of the input data, see example.

name.var.complete  
Boolean. If name.var is supplied with some empty names, name.var.complete allows you to use the initial variable names to complete the graph (from colnames(X)). Default to FALSE.
plotContrib

**title**
A set of characters to indicate the title of the plot. Default value is NULL.

**legend**
Boolean indicating if the legend indicating the group outcomes should be added to the plot. Default value is TRUE.

**legend.color**
A color vector of length the number of group outcomes. See examples.

**legend.title**
A set of characters to indicate the title of the legend. Default value is NULL.

**Details**
This function is a special case of `plotLoadings` and will be discarded in the next update. Please start making the transition to `plotLoadings`, which only requires you to change the name of the call from `plotContrib` to `plotLoadings`.

The contribution of each variable is represented in a barplot where each bar length corresponds to the loading weight (importance) of the feature in (sparse)PLSDA for each component. The loading weight can be positive or negative. The color corresponds to the group in which the feature is most 'abundant'. Note that this type of graphical output is particularly insightful for count microbial data - in that latter case using the method = 'median' is advised.

**Author(s)**
Kim-Anh Le Cao, Benoit Gautier, Florian Rohart, Francois Bartolo

**See Also**
`plotVar`, `cim`, for other variable plots and http://mixOmics.org/graphics for more details.

**Examples**

```r
## object of class 'splsda'
# ------------------------
data(liver.toxicity)
X <- as.matrix(liver.toxicity$gene)
Y <- as.factor(liver.toxicity$treatment[, 4])
splsda.liver <- splsda(X, Y, ncomp = 2, keepX = c(20, 20))

# contribution on comp 1, based on the median.
# Colors indicate the group in which the median expression is maximal
plotContrib(splsda.liver, comp = 1, method = 'median')

# contribution on comp 2, based on median.
# Colors indicate the group in which the median expression is maximal
plotContrib(splsda.liver, comp = 2, method = 'median')

# contribution on comp 2, based on median.
# Colors indicate the group in which the median expression is minimal
plotContrib(splsda.liver, comp = 2, method = 'median', contrib = 'min')

# changing the name to gene names
# if the user input a name.var but names(name.var) is NULL,
# then a warning will be output and assign names of name.var to colnames(X)
```
# this is to make sure we can match the name of the selected variables to the contribution plot.
name.var = liver.toxicity$gene.ID[, 'geneBank']
length(name.var)
plotContrib(splsda.liver, comp = 2, method = 'median', name.var = name.var)

# if names are provided: ok, even when NAs
name.var = liver.toxicity$gene.ID[, 'geneBank']
names(name.var) = rownames(liver.toxicity$gene.ID)
plotContrib(splsda.liver, comp = 2, method = 'median',
name.var = name.var, cex.name = 0.5)

# missing names of some genes? complete with the original names
plotContrib(splsda.liver, comp = 2, method = 'median',
name.var = name.var, cex.name = 0.5, name.var.complete=TRUE)

# look at the contribution (median) for each variable
plot.contrib = plotContrib(splsda.liver, comp = 2, method = 'median', plot = FALSE)
head(plot.contrib$contrib)

# change the title of the legend and title name
plotContrib(splsda.liver, comp = 2, method = 'median', legend.title = 'Time',
title = 'Contribution plot')

# no legend
plotContrib(splsda.liver, comp = 2, method = 'median', legend = FALSE)

# change the color of the legend
plotContrib(splsda.liver, comp = 2, method = 'median', legend.color = c(1:4))

# object 'plsda'
# -----------------
## Not run:
## breast tumors
## ---
data(breast.tumors)
X <- breast.tumors$gene.exp
Y <- breast.tumors$sample$treatment

plsda.breast <- plsda(X, Y, ncomp = 2)

name.var = as.character(breast.tumors$genes$name)
names(name.var) = colnames(X)

# with gene IDs, showing the top 60
plotContrib(plsda.breast, contrib = 'max', comp = 1, method = 'median',
ndisplay = 60,
name.var = name.var,
cex.name = 0.6,
legend.color = color.mixo(1:2))

## End(Not run)
# liver toxicity
# ---
## Not run:
data(liver.toxicity)
X <- liver.toxicity$gene
Y <- liver.toxicity$treatment[, 4]

plsd.liver <- plsd(X, Y, ncomp = 2)
plotIndiv(plsd.liver, ind.names = Y, ellipse = TRUE)

name.var = liver.toxicity$gene.ID[, 'geneBank']
names(name.var) = rownames(liver.toxicity$gene.ID)

plotContribution(plsd.liver, contrib = 'max', comp = 1, method = 'median', ndisplay = 100,
                 name.var = name.var, cex.name = 0.4,
                 legend.color = color.mixo(1:4))

## End(Not run)

# object 'sgccda'
# -------------
## Not run:
data(nutrimouse)
Y = nutrimouse$diet
data = list(gene = nutrimouse$gene, lipid = nutrimouse$lipid)
design = matrix(c(0,1,1,1,0,1,1,1), ncol = 3, nrow = 3, byrow = TRUE)
nutrimouse.sgccda <- wrapper.sgccda(X = data,
                                       Y = Y, design = design,
                                       keepX = list(gene = c(10,10), lipid = c(15,15)),
                                       ncomp = 2, scheme = "centroid",
                                       verbose = FALSE,
                                       bias = FALSE)

plotContribution(nutrimouse.sgccda, block=2)
plotContribution(nutrimouse.sgccda, block="gene")

## End(Not run)

---

plotDiablo  

**Graphical output for the DIABLO framework**

---

**Description**

Function to visualise correlation between components from different data sets.
Usage

plotDiablo(x, ncomp = 1, legend = TRUE, legend.ncol, ...)

Arguments

x
object of class inheriting from "block.splsda".

ncomp
Which component to plot calculated from each data set. Has to be lower than the minimum of object$ncomp.

legend
boolean. Whether the legend should be added. Default is TRUE.

legend.ncol
Number of columns for the legend. Default to min(5,nlevels(x$Y))

... not used

Details

The function uses a plot.data.frame to plot the component ncomp calculated from each data set to visualise whether DIABLO (block.splsda) is successful at maximising the correlation between each data sets' component. The lower triangular panel indicated the Pearson's correlation coefficient, the upper triangular panel the scatter plot.

Author(s)

Amrit Singh

References


See Also

block.splsda and http://www.mixOmics.org/mixDIABLO for more details.

Examples

data('breast.TCGA')
Y = breast.TCGA$data.train$subtype

data = list(mrna = breast.TCGA$data.train$mrna,
            mirna = breast.TCGA$data.train$mirna,
            prot = breast.TCGA$data.train$protein)

# set number of component per data set
ncomp = 3
# set number of variables to select, per component and per data set (arbitrarily set)
list.keepX = list(mrna = rep(20, 3), mirna = rep(10,3), prot = rep(10,3))

# set up a full design where every block is connected
design = matrix(1, ncol = length(data), nrow = length(data),
dimnames = list(names(data), names(data)))
Description

This function provides scatter plots for individuals (experimental units) representation in (sparse)(T)PCA, (regularized)CCA, (sparse)PLS(DA) and (sparse)(R)GCCA(DA).

Usage

## S3 method for class 'pls'
plotIndiv(object, comp = NULL, rep.space = NULL, ind.names = TRUE, group, col.per.group, style = "ggplot2", ellipse = FALSE, ellipse.level = 0.95, centroid = FALSE, star = FALSE, title = NULL, subtitle, legend = FALSE, X.label = NULL, Y.label = NULL, Z.label = NULL, abline = FALSE, xlim = NULL, ylim = NULL, col, cex, pch, pch.levels, alpha = 0.2, axes.box = "box", layout = NULL, size.title = rel(2), size.subtitle = rel(1.5), size.xlabel = rel(1), size.ylabel = rel(1), size.axis = rel(0.8), size.legend = rel(1), size.legend.title = rel(1.1), legend.title = "Legend", legend.title.pch = "Legend", legend.position = "right", point.lwd = 1, background = NULL, ... )

## S3 method for class 'mint.pls'
plotIndiv(object, comp = NULL, study = "global", rep.space = NULL, ind.names = TRUE, group, col.per.group, style = "ggplot2", ellipse = FALSE, ellipse.level = 0.95, centroid = FALSE, star = FALSE, title = NULL, subtitle, legend = FALSE, X.label = NULL, Y.label = NULL, Z.label = NULL, abline = FALSE, xlim = NULL, ylim = NULL, col, cex, pch, layout = NULL, size.title = rel(2), size.subtitle = rel(1.5), size.xlabel = rel(1), size.ylabel = rel(1), size.axis = rel(0.8), size.legend = rel(1), size.legend.title = rel(1.1), legend.title = "Legend", legend.position = "right", point.lwd = 1, ... )

## S3 method for class 'sgcca'
plotIndiv(object, comp = NULL, blocks = NULL, ind.names = TRUE, group, col.per.group, style = "ggplot2", ellipse = FALSE, ellipse.level = 0.95, centroid = FALSE, star = FALSE, title = NULL, subtitle, legend = FALSE, X.label = NULL, Y.label = NULL, Z.label = NULL, abline = FALSE, xlim = NULL, ylim = NULL, col, cex, pch, pch.levels, alpha = 0.2, axes.box = "box", layout = NULL,
Arguments

object object of class inheriting from any mixOmics: PLS, sPLS, PLS-DA, SPLS-DA, rCC, PCA, sPCA, iPCA, iPcaL, splscL, spcaL, ipcaL, sipcaL, rgccaL, sGCCDA, comp integer vector of length two (or three to 3d). The components that will be used on the horizontal and the vertical axis respectively to project the individuals.

rep.space For objects of class "rcc", "pls", "spls", character string, (partially) matching one of "X-variate", "Y-variate", or "XY-variate", determining the subspace to project the individuals. Defaults to "X-variate" "pca" object and for "plsda" objects. For objects of class "pls" and "rcc", defaults, the tree subspaces represent the individuals. For objects of class "rgcca" and "sGCCDA", numerical value indicating the block data set form which to represent the individuals.

blocks integer value of name of a block to be plotted using the GCCA module. See examples.

study Indicates which study-specific outputs to plot. A character vector containing some levels of object$study, "all.partial" to plot all studies or "global" is expected. Default to "global".

ind.names either a character vector of names for the individuals to be plotted, or FALSE for no names. If TRUE, the row names of the first (or second) data matrix is used as names (see Details).

group factor indicating the group membership for each sample, useful for ellipse plots. Coded as default for the supervised methods PLS-DA, SPLS-DA, sGCCDA, but needs to be input for the unsupervised methods PCA, sPCA, iPCA, iPcaL, splscL, spcaL, ipcaL, sipcaL, plsL, splsL, rCC, rGCCDA, style argument to be set to either 'graphics', 'lattice', 'ggplot2' or 3d' for a style of plotting. Default set to 'ggplot2'. See details. 3d is not available for MINT objects.

ellipse boolean indicating if ellipse plots should be plotted. In the non supervised objects PCA, sPCA, iPCA, iPcaL, splscL, spcaL, ipcaL, sipcaL, plsL, splsL, rCC, rGCCDA, sGCCDA ellipse plot is only be plotted if the argument group is provided. In the PLS-DA, SPLS-DA, sGCCDA supervised object, by default the ellipse will be plotted according to the outcome Y.

ellipse.level Numerical value indicating the confidence level of ellipse being plotted when ellipse = TRUE (i.e. the size of the ellipse). The default is set to 0.95, for a 95% region.
Plot individual graphs using the following arguments:

- **centroid**: boolean indicating whether centroid points should be plotted. In the non-supervised objects PCA, sPCA, IPCA, sIPCA, PLS, sPLS, rCC, rGCCCA, sGCCCA, the centroid will only be plotted if the argument group is provided. The centroid will be calculated based on the group categories. In the supervised objects PLS-DA, SPLS-DA, sGCCDA, the centroid will be calculated according to the outcome Y.

- **star**: boolean indicating whether a star plot should be plotted, with arrows starting from the centroid (see argument centroid, and ending for each sample belonging to each group or outcome. In the non-supervised objects PCA, sPCA, IPCA, sIPCA, PLS, sPLS, rCC, rGCCCA, the star plot is only be plotted if the argument group is provided. In the supervised objects PLS-DA, SPLS-DA, sGCCDA, the star plot is plotted according to the outcome Y.

- **title**: set of characters indicating the title plot.
- **subtitle**: subtitle for each plot, only used when several block or study are plotted.
- **legend**: boolean. Whether the legend should be added. Default is FALSE.

- **x.axis**: x axis titles.
- **y.axis**: y axis titles.
- **z.axis**: z axis titles (when style = '3d').
- **abline**: should the vertical and horizontal line through the center be plotted? Default set to FALSE.
- **xlim, ylim**: numeric list of vectors of length 2 and length =length(blocks), giving the x and y coordinates ranges.
- **col**: character (or symbol) color to be used, possibly vector.
- **cex**: numeric character (or symbol) expansion, possibly vector.
- **pch**: plot character. A character string or a vector of single characters or integers. See `points` for all alternatives.
- **pch.levels**: Only used when pch is different from col or col.per.group, ie when pch creates a second factor. Only used for the legend.
- **alpha**: Semi-transparent colors (0 < 'alpha' < 1)
- **axes.box**: for style '3d', argument to be set to either 'axes', 'box', 'bbox' or 'all', defining the shape of the box.
- **layout**: layout parameter passed to mfrow. Only used when study is not "global"
- **size.title**: size of the title
- **size.subtitle**: size of the subtitle
- **size.xlabel**: size of xlabel
- **size.ylabel**: size of ylabel
- **size.axis**: size of the axis
- **size.legend**: size of the legend
- **size.legend.title**: size of the legend title
- **legend.title**: title of the legend
plotIndiv

- **legend.title.pch**: title of the second legend created by pch, if any.
- **legend.position**: position of the legend, one of "bottom", "left", "top" and "right".
- **point.lwd**: lwd of the points, used when ind.names = FALSE
- **background**: color the background by the predicted class, see background.predict
- **...**: external arguments or type par can be added with style = 'graphics'

### Details

plotIndiv method makes scatter plot for individuals representation depending on the subspace of projection. Each point corresponds to an individual.

If ind.names=TRUE and row names is NULL, then ind.names=1:n, where n is the number of individuals. Also, if pch is an input, then ind.names is set to FALSE as we do not show both names and shapes.

plotIndiv can have a two layers legend. This is especially convenient when you have two grouping factors, such as a gender effect and a study effect, and you want to highlight both simultaneously on the graphical output. A first layer is coded by the group factor, the second by the pch argument. When pch is missing, a single layer legend is shown. If the group factor is missing, the col argument is used to create the grouping factor group. When a second grouping factor is needed and added via pch, pch needs to be a vector of length the number of samples. In the case where pch is a vector or length the number of groups, then we consider that the user wants a different pch for each level of group. This leads to a single layer legend and we merge col and pch. In the similar case where pch is a single value, then this value is used to represent all samples. See examples below for object of class plsda and splsda.

In the specific case of a single ‘omics supervised model (plsda, splsda), users can overlay prediction results to sample plots in order to visualise the prediction areas of each class, via the background input parameter. Note that this functionality is only available for models with less than 2 components as the surfaces obtained for higher order components cannot be projected onto a 2D representation in a meaningful way. For more details, see background.predict.

For customized plots (i.e. adding points, text), use the style = 'graphics' (default is ggplot2).

Note: the ellipse options were borrowed from the ellipse.

### Author(s)

Ignacio Gonzalez, Benoit Gautier, Francois Bartolo, Florian Rohart

### See Also

- text, background.predict, points and http://mixOmics.org/graphics for more details.

### Examples

```r
# plot of individuals for objects of class 'rcc'
# ---------------------------------------------
data(nutrimouse)
X <- nutrimouse$lipid
```
Y <- nutrimouse$gene
nutri.res <- rcc(X, Y, ncomp = 3, lambda1 = 0.064, lambda2 = 0.008)

# default, only in the X space
plotIndiv(nutri.res)

## Not run:
# ellipse with respect to genotype in the XY space,
# names also indicate genotype
plotIndiv(nutri.res, rep.space = 'XY-variate',
    ellipse = TRUE, ellipse.level = 0.9,
    group = nutrimouse$genotype, ind.names = nutrimouse$genotype)

# ellipse with respect to genotype in the XY space, with legend
plotIndiv(nutri.res, rep.space = 'XY-variate', group = nutrimouse$genotype,
    legend = TRUE)

# lattice style
plotIndiv(nutri.res, rep.space = 'XY-variate', group = nutrimouse$genotype,
    legend = TRUE, style = 'lattice')

# classic style, in the Y space
plotIndiv(nutri.res, rep.space = 'Y-variate', group = nutrimouse$genotype,
    legend = TRUE, style = 'graphics')

## Not run

## plot of individuals for objects of class 'pls' or 'spls'
# ----------------------------------------------
data(liver.toxicity)
X <- liver.toxicity$gene
Y <- liver.toxicity$clinic
toxicity.spls <- spls(X, Y, ncomp = 3, keepX = c(50, 50, 50),
    keepY = c(10, 10, 10))

# default
plotIndiv(toxicity.spls)

## Not run:
# two layers legend: a first grouping with Time.Group and 'group'
# and a second with Dose.Group and 'pch'
plotIndiv(toxicity.spls, rep.space = "X-variate", ind.name = FALSE,
    group = liver.toxicity$treatment[, 'Time.Group'],
    pch = as.numeric(factor(liver.toxicity$treatment$Dose.Group)),
    pch.levels = liver.toxicity$treatment$Dose.Group,
    legend = TRUE)

## End(Not run)

# indicating the centroid
plotIndiv(toxicity.spls, rep.space = 'X-variate', ind.names = FALSE,
group = liver.toxicity$treatment[, 'Time.Group'], centroid = TRUE)

# indicating the star and centroid
plotIndiv(toxicity.spls, rep.space = 'X-variate', ind.names = FALSE,
         group = liver.toxicity$treatment[, 'Time.Group'], centroid = TRUE, star = TRUE)

# indicating the star and ellipse
plotIndiv(toxicity.spls, rep.space = 'X-variate', ind.names = FALSE,
         group = liver.toxicity$treatment[, 'Time.Group'], centroid = TRUE,
         star = TRUE, ellipse = TRUE)

# in the Y space, colors indicate time of necropsy, text is the dose
plotIndiv(toxicity.spls, rep.space = 'Y-variate',
         group = liver.toxicity$treatment[, 'Time.Group'],
         ind.names = liver.toxicity$treatment[, 'Dose.Group'],
         legend = TRUE)

## plot of individuals for objects of class 'plsd'a or 'splsda'
# -----------------------------------------------
data(breast.tumors)
X <- breast.tumors$gene.exp
Y <- breast.tumors$sample$treatment
splsda.breast <- splsda(X, Y, keepX=c(10,10), ncomp=2)

# default option: note the outcome color is included by default!
plotIndiv(splsda.breast)

# also check ?background.predict for to visualise the prediction
# area with a plsd'a or splsda object!

## Not run:
# default option with no ind name: pch and color are set automatically
plotIndiv(splsda.breast, ind.names = FALSE, comp = c(1, 2))

# default option with no ind name: pch and color are set automatically, with legend
plotIndiv(splsda.breast, ind.names = FALSE, comp = c(1, 2), legend = TRUE)

# trying the different styles
plotIndiv(splsda.breast, ind.names = TRUE, comp = c(1, 2),
         ellipse = TRUE, style = "ggplot2", cex = c(1, 1))
plotIndiv(splsda.breast, ind.names = TRUE, comp = c(1, 2),
         ellipse = TRUE, style = "lattice", cex = c(1, 1))

# changing pch of the two groups
plotIndiv(splsda.breast, ind.names = FALSE, comp = c(1, 2),
         pch = c(15,16), legend = TRUE)
# creating a second grouping factor with a pch of length 3, # which is recycled to obtain a vector of length n
plotIndiv(splsda.breast, ind.names = FALSE, comp = c(1, 2),
pch = c(15, 16, 17), legend = TRUE)

#same thing as
pch.indiv = c(rep(15:17, 15), 15, 16) # length n
plotIndiv(splsda.breast, ind.names = FALSE, comp = c(1, 2),
pch = pch.indiv, legend = TRUE)

# change the names of the second legend with pch.levels
plotIndiv(splsda.breast, ind.names = FALSE, comp = c(1, 2),
pch = 15:17, pch.levels = c("a","b","c"), legend = TRUE)

## End(Not run)

## plot of individuals for objects of class 'mint.plsda' or 'mint.splsda'
# -----------------------------
da(stemcells)
res = mint.splsda(X = stemcells$gene, Y = stemcells$celltype, ncomp = 2, keepX = c(10, 5),
study = stemcells$study)
plotIndiv(res)

## Not run:
#plot study-specific outputs for all studies
plotIndiv(res, study = "all.partial")

#plot study-specific outputs for study "2"
plotIndiv(res, study = "2")

## End(Not run)

## variable representation for objects of class 'sgcca' (or 'rgcca')
# -----------------------------
## Not run:
da(nutrimouse)
Y = unmap(nutrimouse$diet)
da = list(gene = nutrimouse$gene, lipid = nutrimouse$lipid, Y = Y)
design1 = matrix(c(0, 1, 1, 1, 0, 1, 1, 1, 0), ncol = 3, nrow = 3, byrow = TRUE)
nutrimouse.sgcca <- wrapper.sgcca(X = data,
design = design1,
penalty = c(0.3, 0.5, 1),
ncomp = 3,
scheme = "horst",
verbose = FALSE,
bias = FALSE)

# default style: one panel for each block
plotIndiv(nutrimouse.sgcca)
# for the block 'lipid' with ellipse plots and legend, different styles
plotIndiv(nutrimouse.sgcca, group = nutrimouse$diet, legend = TRUE, ellipse = TRUE, ellipse.level = 0.5, blocks = "lipid", title = 'my plot')
plotIndiv(nutrimouse.sgcca, style = "lattice", group = nutrimouse$diet, legend = TRUE, ellipse = TRUE, ellipse.level = 0.5, blocks = "lipid", title = 'my plot')
plotIndiv(nutrimouse.sgcca, style = "graphics", group = nutrimouse$diet, legend = TRUE, ellipse = TRUE, ellipse.level = 0.5, blocks = "lipid", title = 'my plot')

## End(Not run)

## variable representation for objects of class 'sgccda'
## -----------------------------------------------
## Not run:
# Note: the code differs from above as we use a 'supervised' GCCA analysis
data(nutrimouse)
Y = nutrimouse$diet
data = list(gene = nutrimouse$gene, lipid = nutrimouse$lipid)
design1 = matrix(c(0,1,0,1), ncol = 2, nrow = 2, byrow = TRUE)
nutrimouse.sgccdal <- wrapper.sgcca(X = data, Y = Y, design = design1, ncomp = 2, keepX = list(gene = c(10,10), lipid = c(15,15)), scheme = "centroid", verbose = FALSE, bias = FALSE)

# plotIndiv
# ---------

# displaying all blocks. bu default colors correspond to outcome Y
plotIndiv(nutrimouse.sgccdal)

# displaying only 2 blocks
plotIndiv(nutrimouse.sgccdal, blocks = c(1,2), group = nutrimouse$diet)

# with some ellipse, legend and title
plotIndiv(nutrimouse.sgccdal, blocks = c(1,2), group = nutrimouse$diet, ellipse = TRUE, legend = TRUE, title = 'my sample plot')

## End(Not run)
Description

This function provides a horizontal bar plot to visualise loading vectors. For discriminant analysis, it provides visualisation of highest or lowest mean/median value of the variables with color code corresponding to the outcome of interest.

Usage

## S3 method for class 'pls'
plotLoadings(object, block, comp = 1, col = NULL, ndisplay = NULL, size.name = 0.7, name.var = NULL, name.var.complete = FALSE, title = NULL, subtitle, size.title = rel(2), size.subtitle = rel(1.5), layout = NULL, border = NA, xlim = NULL, ... )

## S3 method for class 'mint.pls'
plotLoadings(object, study = "global", comp = 1, col = NULL, ndisplay = NULL, size.name = 0.7, name.var = NULL, name.var.complete = FALSE, title = NULL, subtitle, size.title = rel(1.8), size.subtitle = rel(1.4), layout = NULL, border = NA, xlim = NULL, ... )

## S3 method for class 'plsda'
plotLoadings(object, contrib, method = "mean", block, comp = 1, plot = TRUE, show.ties = TRUE, col.ties="white", ndisplay = NULL, size.name = 0.7, size.legend = 0.8, name.var=NULL, name.var.complete=FALSE, title = NULL, subtitle, size.title = rel(1.8), size.subtitle = rel(1.4), legend = TRUE, legend.color = NULL, legend.title = 'Outcome', layout = NULL, border = NA, xlim = NULL, ... )

## S3 method for class 'mint.plsda'
plotLoadings(object, contrib = NULL, method = "mean", study = "global", comp = 1, plot = TRUE, show.ties = TRUE, col.ties = "white", ndisplay = NULL, size.name = 0.7, size.legend = 0.8, name.var = NULL, name.var.complete = FALSE, title = NULL, subtitle, size.title = rel(1.8), size.subtitle = rel(1.4), legend = TRUE, legend.color = NULL, legend.title = 'Outcome', layout = NULL, border = NA, xlim = NULL, ... )

Arguments

- **object**: object,
- **contrib**: a character set to 'max' or 'min' indicating if the color of the bar should correspond to the group with the maximal or minimal expression levels / abundance.
- **method**: a character set to 'mean' or 'median' indicating the criterion to assess the contribution. We recommend using median in the case of count or skewed data.
- **study**: Indicates which study are to be plotted. A character vector containing some levels of object$study, "all.partial" to plot all studies or "global" is expected.
- **block**: A single value indicating which block to consider in a sgccda object.
- **comp**: integer value indicating the component of interest from the object.
**Details**

The contribution of each variable for each component (depending on the object) is represented in a barplot where each bar length corresponds to the loading weight (importance) of the feature. The loading weight can be positive or negative.

For discriminant analysis, the color corresponds to the group in which the feature is most ‘abundant’. Note that this type of graphical output is particularly insightful for count microbial data.
that latter case using the method = 'median' is advised. Note also that if the parameter contrib is not provided, plots are white.

For MINT analysis, study="global" plots the global loadings while partial loadings are plotted when study is a level of object$study. Since variable selection in MINT is performed at the global level, only the selected variables are plotted for the partial loadings even if the partial loadings are not sparse. See references. Importantly for multi plots, the legend accounts for one subplot in the layout design.

Author(s)

Florian Rohart, Kim-Anh Le Cao, Benoit Gautier

References


See Also

`pls, spls, plsda, splsda, mint.pls, mint.spls, mint.plsda, mint.splsda, block.pls, block.spls, block.plsda, block.splsda, mint.block.pls, mint.block.spls, mint.block.plsda, mint.block.splsda`

Examples

```r
## object of class 'spls'
# -------------------------
data(liver.toxicity)
X = liver.toxicity$gene
Y = liver.toxicity$clinic

toxicity.spls = spls(X, Y, ncomp = 2, keepX = c(50, 50),
keepY = c(10, 10))

plotLoadings(toxicity.spls)

# with xlim
xlim = matrix(c(-0.1, 0.3, -0.4, 0.6), nrow = 2, byrow = TRUE)
plotLoadings(toxicity.spls, xlim = xlim)
```
## Code Snippet

```r
# object of class 'splsda'
# -------------------------------
data(liver.toxicity)
X = as.matrix(liver.toxicity$gene)
Y = as.factor(liver.toxicity$treatment[, 4])

splsda.liver = splsda(X, Y, ncomp = 2, keepX = c(20, 20))

# contribution on comp 1, based on the median.
# Colors indicate the group in which the median expression is maximal
plotLoadings(splsda.liver, comp = 1, method = 'median')

# contribution on comp 2, based on median.
# Colors indicate the group in which the median expression is maximal
plotLoadings(splsda.liver, comp = 2, method = 'median', contrib = "max")

# changing the name to gene names
# if the user input a name.var but names(name.var) is NULL,
# then a warning will be output and assign names of name.var to colnames(X)
# this is to make sure we can match the name of the selected variables to the contribution plot.
names(name.var) = rownames(liver.toxicity$gene.ID[, 'geneBank'])
plotLoadings(splsda.liver, comp = 2, method = 'median', name.var = name.var,
            title = "Liver data", contrib = "max")

# if names are provided: ok, even when NAS
name.var = liver.toxicity$gene.ID[, 'geneBank']
names(name.var) = rownames(liver.toxicity$gene.ID)
plotLoadings(splsda.liver, comp = 2, method = 'median',
            name.var = name.var, size.name = 0.5, contrib = "max")

# missing names of some genes? complete with the original names
plotLoadings(splsda.liver, comp = 2, method = 'median',
            name.var = name.var, size.name = 0.5, complete.name.var=TRUE, contrib = "max")

# look at the contribution (median) for each variable
plot.contrib = plotLoadings(splsda.liver, comp = 2, method = 'median', plot = FALSE, contrib = "max")
head(plot.contrib$contrib)

# change the title of the legend and title name
plotLoadings(splsda.liver, comp = 2, method = 'median', legend.title = 'Time',
            title = 'Contribution plot', contrib = "max")

# no legend
plotLoadings(splsda.liver, comp = 2, method = 'median', legend = FALSE, contrib = "max")

# change the color of the legend
```
plotLoadings(splsda.liver, comp = 2, method = 'median', legend.color = c(1:4), contrib = "max")

# object 'splsda multilevel'
# --------------
## Not run:
data(vac18)
X = vac18$genes
Y = vac18$stimulation
# sample indicates the repeated measurements
sample = vac18$sample
stimul = vac18$stimulation

# multilevel sPLS-DA model
res.nlevel = splsda(X, Y = stimul, ncomp = 3, multilevel = sample,
keepX = c(30, 137, 123))

name.var = vac18$tab.prob.gene[, 'Gene']
names(name.var) = colnames(X)

plotLoadings(res.nlevel, comp = 2, method = 'median', legend.title = 'Stimu',
name.var = name.var, size.name = 0.2, contrib = "max")

# too many transcripts? only output the top ones
plotLoadings(res.nlevel, comp = 2, method = 'median', legend.title = 'Stimu',
name.var = name.var, size.name = 0.5, ndisplay = 60, contrib = "max")

## End(Not run)

# object 'plsda'
# ---------------
## Not run:
# breast tumors
# ---
data(breast.tumors)
X = breast.tumors$gene.exp
Y = breast.tumors$sample$treatment

plsda.breast = plsda(X, Y, ncomp = 2)

name.var = as.character(breast.tumors$genes$name)
names(name.var) = colnames(X)

# with gene IDs, showing the top 60
plotLoadings(plsda.breast, contrib = 'max', comp = 1, method = 'median',
ndisplay = 60,
name.var = name.var,
size.name = 0.6,
legend.color = color.mixo(1:2))
## End(Not run)

# liver toxicity
# ---
## Not run:
data(liver.toxicity)
X = liver.toxicity$gene
Y = liver.toxicity$treatment[, 4]

plsd.liver = plsd(X, Y, ncomp = 2)
plotIndiv(plsd.liver, ind.names = Y, ellipse = TRUE)

name.var = liver.toxicity$gene.ID[, 'geneBank']
names(name.var) = rownames(liver.toxicity$gene.ID)

plotLoadings(plsd.liver, contrib = 'max', comp = 1, method = 'median', ndisplay = 100,
name.var = name.var, size.name = 0.4,
legend.color = color.mixo(1:4))

## End(Not run)

# object 'sgccda'
# ---------------
## Not run:
data(nutrimouse)
Y = nutrimouse$diet
data = list(gene = nutrimouse$gene, lipid = nutrimouse$lipid)
design = matrix(c(0,1,1,1,0,1,1,0), ncol = 3, nrow = 3, byrow = TRUE)

nutrimouse.sgccda = wrapper.sgccda(X = data,
Y = Y,
design = design,
keepX = list(gene = c(10,10), lipid = c(15,15)),
ncomp = 2,
scheme = "centroid",
verbose = FALSE,
bias = FALSE)

plotLoadings(nutrimouse.sgccda,block=2)
plotLoadings(nutrimouse.sgccda,block="gene")

## End(Not run)

# object 'mint.splsda'
# -------------------
data(stemcells)
data = stemcells$gene
type.id = stemcells$celltype
exp = stemcells$study

res = mint.splsda(X = data, Y = type.id, ncomp = 3, keepX = c(10,5,15), study = exp)
plotLoadings(res)
plotLoadings(res, contrib = "max")
plotLoadings(res, contrib = "min", study = 1:4, comp=2)

# combining different plots by setting a layout of 2 rows and 4 columns.
# Note that the legend accounts for a subplot so 4 columns instead of 2.
plotLoadings(res, contrib="min", study=c(1,2,3), comp=2, layout = c(2,4))
plotLoadings(res, contrib="min", study="global", comp=2)

plotVar

Description

This function provides variables representation for (regularized) CCA, (sparse) PLS regression,
PCA and (sparse) Regularized generalised CCA.

Usage

plotVar(object,
comp = NULL,
comp.select = comp,
plot = TRUE,
var.names = NULL,
blocks = NULL, # to choose which block data to plot, when using GCCA module
X.label = NULL,
Y.label = NULL,
Z.label = NULL,
abline = TRUE,
col,
cex,
pch,
font,
cutoff = 0,
rad.in = 0.5,
title="Correlation Circle Plots",
legend = FALSE,
style="ggplot2", # can choose between graphics, 3d, lattice or ggplot2,
overlap = TRUE,
axes.box = "all",
label.axes.box = "both")
Arguments

object

object of class inheriting from "rcc", "pls", "plsda", "spls", "splsda", "pca" or "spca".

comp

integer vector of length two. The components that will be used on the horizontal and the vertical axis respectively to project the variables. By default, comp=c(1,2) except when style='3d', comp=c(1:3)

comp.select

for the sparse versions, an input vector indicating the components on which the variables were selected. Only those selected variables are displayed. By default, comp.select=comp

plot

if TRUE (the default) then a plot is produced. If not, the summaries which the plots are based on are returned.

var.names

either a character vector of names for the variables to be plotted, or FALSE for no names. If TRUE, the col names of the first (or second) data matrix is used as names.

blocks

for an object of class "rgcca" or "sgcca", a numerical vector indicating the block variables to display.

X.label

x axis titles.

Y.label

y axis titles.

Z.label

z axis titles (when style = '3d').

abline

should the vertical and horizontal line through the center be plotted? Default set to FALSE

col

character or integer vector of colors for plotted character and symbols, can be of length 2 (one for each data set) or of length (p+q) (i.e. the total number of variables). See Details.

cex

numeric vector of character expansion sizes for the plotted character and symbols, can be of length 2 (one for each data set) or of length (p+q) (i.e. the total number of variables).

pch

plot character. A vector of single characters or integers, can be of length 2 (one for each data set) or of length (p+q) (i.e. the total number of variables). See points for all alternatives.

font

numeric vector of font to be used, can be of length 2 (one for each data set) or of length (p+q) (i.e. the total number of variables). See par for details.

cutoff

numeric between 0 and 1. Variables with correlations below this cutoff in absolute value are not plotted (see Details).

rad.in

numeric between 0 and 1, the radius of the inner circle. Defaults to 0.5.

title

character indicating the title plot.

legend

boolean. Whether the legend should be added. Default is TRUE.

style

argument to be set to either 'graphics', 'lattice', 'ggplot2' or '3d' for a style of plotting.

overlap

boolean. Whether the variables should be plotted in one single figure. Default is TRUE.
plotVar

axes.box for style '3d', argument to be set to either 'axes', 'box', 'bbox' or 'all', defining the shape of the box.

label.axes.box for style '3d', argument to be set to either 'axes', 'box', 'both', indicating which labels to print.

Details

plotVar produce a "correlation circle", i.e. the correlations between each variable and the selected components are plotted as scatter plot, with concentric circles of radius one et radius given by rad.in. Each point corresponds to a variable. For (regularized) CCA the components correspond to the equiangular vector between X- and Y-variates. For (sparse) PLS regression mode the components correspond to the X-variates. If mode is canonical, the components for X and Y variables correspond to the X- and Y-variates respectively.

For plsda and splsda objects, only the X variables are represented.

For spls and splsda objects, only the X and Y variables selected on dimensions comp are represented.

The arguments col, pch, cex and font can be either vectors of length two or a list with two vector components of length p and q respectively, where p is the number of X-variables and q is the number of Y-variables. In the first case, the first and second component of the vector determine the graphics attributes for the X- and Y-variables respectively. Otherwise, multiple arguments values can be specified so that each point (variable) can be given its own graphic attributes. In this case, the first component of the list correspond to the X attributs and the second component correspond to the Y attributs. Default values exist for this arguments.

Value

A list containing the following components:

x a vector of coordinates of the variables on the x-axis.
y a vector of coordinates of the variables on the y-axis.
Block the data block name each variable belongs to.
names the name of each variable, matching their coordinates values.

Author(s)

Ignacio Gonzalez, Kim-Anh Le Cao, Benoit Gautier, Florian Rohart, Francois Bartolo.

References


See Also

cim, network, par and http://www.mixOmics.org for more details.
Examples

```r
## variable representation for objects of class 'rcc'
# -----------------------------------------------
data(nutrimouse)
X <- nutrimouse$lipid
Y <- nutrimouse$gene
nutri.res <- rcc(X, Y, ncomp = 3, lambda1 = 0.064, lambda2 = 0.008)

plotVar(nutri.res) #(default)

## Not run:
plotVar(nutri.res, comp = c(1,3), cutoff = 0.5)

## End(Not run)

## variable representation for objects of class 'pls' or 'spls'
# -----------------------------------------------
data(liver.toxicity)
X <- liver.toxicity$gene
Y <- liver.toxicity$clinic
toxicity.spls <- spls(X, Y, ncomp = 3, keepX = c(50, 50, 50),
                      keepY = c(10, 10, 10))

plotVar(toxicity.spls, cex = c(1,0.8))

## variable representation for objects of class 'splsda'
# -----------------------------------------------
## Not run:
data(liver.toxicity)
X <- liver.toxicity$gene
Y <- as.factor(liver.toxicity$treatment[, 4])

ncomp <- 2
keepX <- rep(20, ncomp)
splsda.liver <- splsda(X, Y, ncomp = ncomp, keepX = keepX)
plotVar(splsda.liver)

## End(Not run)

## variable representation for objects of class 'sgcca' (or 'rgcca')
# -----------------------------------------------
## see example in ??wrapper.sgcca
data(nutrimouse)
# need to unmap the Y factor diet
Y = unmap(nutrimouse$diet)
# set up the data as list
data = list(gene = nutrimouse$gene, lipid = nutrimouse$lipid, Y = Y)

# set up the design matrix:
# with this design, gene expression and lipids are connected to the diet factor
# design = matrix(c(0,0,1,
# plotVar

```
# 0,0,1,
# 1,1,0), ncol = 3, nrow = 3, byrow = TRUE)

# with this design, gene expression and lipids are connected to the diet factor
# and gene expression and lipids are also connected
design = matrix(c(0,1,1,
1,0,1,
1,1,0), ncol = 3, nrow = 3, byrow = TRUE)

#note: the penalty parameters will need to be tuned
wrap.result.sgccca = wrapper.sgccca(X = data, design = design, penalty = c(.3,.3, 1),
                  ncomp = 2,
                  scheme = "centroid", verbose = FALSE)

#variables selected on component 1 for each block
selectVar(wrap.result.sgccca, comp = 1, block = c(1,2))$'gene'$name
selectVar(wrap.result.sgccca, comp = 1, block = c(1,2))$'lipid'$name

#variables selected on component 2 for each block
selectVar(wrap.result.sgccca, comp = 2, block = c(1,2))$'gene'$name
selectVar(wrap.result.sgccca, comp = 2, block = c(1,2))$'lipid'$name

plotVar(wrap.result.sgccca, comp = c(1,2), block = c(1,2), comp.select = c(1,1),
        title = c('Variables selected on component 1 only'))

## Not run:
plotVar(wrap.result.sgccca, comp = c(1,2), block = c(1,2), comp.select = c(2,2),
        title = c('Variables selected on component 2 only'))

  # -> this one shows the variables selected on both components
plotVar(wrap.result.sgccca, comp = c(1,2), block = c(1,2),
        title = c('Variables selected on components 1 and 2'))

## End(Not run)
## variable representation for objects of class 'rgcca'
# ---------------------------------------------------------------
## Not run:
data(nutrimouse)
# need to unmap Y for an unsupervised analysis, where Y is included as a data block in data
Y = unmap(nutrimouse$diet)

data = list(gene = nutrimouse$gene, lipid = nutrimouse$lipid, Y = Y)

# with this design, all blocks are connected
design = matrix(c(0,1,1,1,0,1,1,1,0), ncol = 3, nrow = 3,
                byrow = TRUE, dimnames = list(names(data), names(data)))

nutrimouse.rgcca <- wrapper.rgcca(X = data,
                                   design = design,
                                   tau = "optimal",
                                   ncomp = 2,
                                   scheme = "centroid",
                                   ...)```
pls

Partial Least Squares (PLS) Regression

Description

Function to perform Partial Least Squares (PLS) regression.

Usage

pls(X, 
Y, 
ncomp = 2, 
scale = TRUE, 
mode = c("regression", "canonical", "invariant", "classic"), 
tol = 1e-06, 
max.iter = 100, 
near.zero.var = FALSE, 
logratio="none", 
multilevel=NULL)
Arguments

- **X**: numeric matrix of predictors. NAs are allowed.
- **Y**: numeric vector or matrix of responses (for multi-response models). NAs are allowed.
- **ncomp**: the number of components to include in the model. Default to 2.
- **scale**: boolean. If scale = TRUE, each block is standardized to zero means and unit variances (default: TRUE).
- **mode**: character string. What type of algorithm to use, (partially) matching one of "regression", "canonical", "invariant" or "classic". See Details.
- **tol**: Convergence stopping value.
- **max.iter**: integer, the maximum number of iterations.
- **near.zero.var**: boolean, see the internal nearZeroVar function (should be set to TRUE in particular for data with many zero values). Setting this argument to FALSE (when appropriate) will speed up the computations. Default value is FALSE.
- **logratio**: one of (‘none’, ‘CLR’). Default to ‘none’.
- **multilevel**: Design matrix for repeated measurement analysis, where multilevel decomposition is required. For a one factor decomposition, the repeated measures on each individual, i.e. the individuals ID is input as the first column. For a 2 level factor decomposition then 2nd AND 3rd columns indicate those factors. See examples in ?spls).

Details

- `pls` function fit PLS models with 1,...,ncomp components. Multi-response models are fully supported. The X and Y datasets can contain missing values.

The type of algorithm to use is specified with the mode argument. Four PLS algorithms are available: PLS regression ("regression"), PLS canonical analysis ("canonical"), redundancy analysis ("invariant") and the classical PLS algorithm ("classic") (see References). Different modes relate on how the Y matrix is deflated across the iterations of the algorithms - i.e. the different components.

- **Regression mode**: the Y matrix is deflated with respect to the information extracted/modelled from the local regression on X. Here the goal is to predict Y from X (Y and X play an assymmetric role). Consequently the latent variables computed to predict Y from X are different from those computed to predict X from Y.
- **Canonical mode**: the Y matrix is deflated to the information extracted/modelled from the local regression on Y. Here X and Y play a symetric role and the goal is similar to a Canonical Correlation type of analysis.
- **Invariant mode**: the Y matrix is not deflated - Classic mode: is similar to a regression mode. It gives identical results for the variates and loadings associated to the X data set, but differences for the loadings vectors associated to the Y data set (different normalisations are used). Classic mode is the PLS2 model as defined by Tenenhaus (1998), Chap 9. Note that in all cases the results are the same on the first component as deflation only starts after component 1.

The estimation of the missing values can be performed by the reconstitution of the data matrix using the nipals function. Otherwise, missing values are handled by casewise deletion in the pls function without having to delete the rows with missing data.

- **logratio transform** and multilevel analysis are performed sequentially as internal pre-processing step, through `logratio.transfo` and `withinVariation` respectively.
Value

pls returns an object of class "pls", a list that contains the following components:

- **X** the centered and standardized original predictor matrix.
- **Y** the centered and standardized original response vector or matrix.
- **ncomp** the number of components included in the model.
- **mode** the algorithm used to fit the model.
- **variates** list containing the variates.
- **loadings** list containing the estimated loadings for the X and Y variates.
- **names** list containing the names to be used for individuals and variables.
- **tol** the tolerance used in the iterative algorithm, used for subsequent S3 methods.
- **iter** Number of iterations of the algorithm for each component.
- **max.iter** the maximum number of iterations, used for subsequent S3 methods.
- **nzv** list containing the zero- or near-zero predictors information.
- **scale** whether scaling was applied per predictor.
- **logratio** whether log ratio transformation for relative proportion data was applied, and if so, which type of transformation.
- **explained_variance** amount of variance explained per component (note that contrary to PCA, this amount may not decrease as the aim of the method is not to maximise the variance, but the covariance between data sets).
- **input.X** numeric matrix of predictors in X that was input, before any saling / logratio / multilevel transformation.
- **mat.c** matrix of coefficients from the regression of X / residual matrices X on the X-variates, to be used internally by predict.
- **defl.matrix** residual matrices X for each dimension.

Author(s)

Sebastien Dejean and Ignacio Gonzalez and Kim-Anh Le Cao.

References


See Also

`spls, summary, plotIndiv, plotVar, predict, perf` and http://www.mixOmics.org for more details.
Examples

```
data(linnerud)
X <- linnerud$exercise
Y <- linnerud$physiological
linn.pls <- pls(X, Y, mode = "classic")

data(liver.toxicity)
X <- liver.toxicity$gene
Y <- liver.toxicity$clinic

toxicity.pls <- pls(X, Y, ncomp = 3)
```

```R
plsda

Partial Least Squares Discriminant Analysis (PLS-DA).
```

Description

Function to perform standard Partial Least Squares regression to classify samples.

Usage

```
plsda(X, 
Y, 
ncomp = 2, 
scale = TRUE, 
mode = c("regression", "canonical", "invariant", "classic"), 
tol = 1e-06, 
max.iter = 100, 
near.zero.var = FALSE, 
logratio="none", # one of "none", "CLR"
multilevel=NULL)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>numeric matrix of predictors. NAs are allowed.</td>
</tr>
<tr>
<td>Y</td>
<td>a factor or a class vector for the discrete outcome.</td>
</tr>
<tr>
<td>ncomp</td>
<td>the number of components to include in the model. Default to 2.</td>
</tr>
<tr>
<td>scale</td>
<td>boolean. If scale = TRUE, each block is standardized to zero means and unit variances (default: TRUE)</td>
</tr>
<tr>
<td>mode</td>
<td>character string. What type of algorithm to use, (partially) matching one of &quot;regression&quot;, &quot;canonical&quot;, &quot;invariant&quot; or &quot;classic&quot;. See Details.</td>
</tr>
<tr>
<td>tol</td>
<td>Convergence stopping value.</td>
</tr>
<tr>
<td>max.iter</td>
<td>integer, the maximum number of iterations.</td>
</tr>
<tr>
<td>near.zero.var</td>
<td>boolean, see the internal nearZeroVar function (should be set to TRUE in particular for data with many zero values). Setting this argument to FALSE (when appropriate) will speed up the computations. Default value is FALSE</td>
</tr>
</tbody>
</table>
logratio one of ('none','CLR') specifies the log ratio transformation to deal with compositional values that may arise from specific normalisation in sequencing data. Default to 'none'
multilevel sample information for multilevel decomposition for repeated measurements. A numeric matrix or data frame indicating the repeated measures on each individual, i.e. the individuals ID. See examples in ?splsda.

Details

plsda function fit PLS models with 1,...,ncomp components to the factor or class vector Y. The appropriate indicator matrix is created.

logratio transform and multilevel analysis are performed sequentially as internal pre-processing step, through logratioNtransfo and withinVariation respectively.

Logratio can only be applied if the data do not contain any 0 value (for count data, we thus advise the normalise raw data with a 1 offset).

More details about the PLS modes in ?pls.

Value

plsda returns an object of class "plsda", a list that contains the following components:

X the centered and standardized original predictor matrix.
Y the centered and standardized indicator response vector or matrix.
ind.mat the indicator matrix.
ncomp the number of components included in the model.
variates list containing the X and Y variates.
loadings list containing the estimated loadings for the variates.
names list containing the names to be used for individuals and variables.
nzv list containing the zero- or near-zero predictors information.
tol the tolerance used in the iterative algorithm, used for subsequent S3 methods
max.iter the maximum number of iterations, used for subsequent S3 methods
iter Number of iterations of the algorithm for each component
explained_variance amount of variance explained per component (note that contrary to PCA, this amount may not decrease as the aim of the method is not to maximise the variance, but the covariance between X and the dummy matrix Y).
mat.c matrix of coefficients from the regression of X / residual matrices X on the X-variates, to be used internally by predict.
defl.matrix residual matrices X for each dimension.

Author(s)

Ignacio Gonzalez, Kim-Anh Le Cao.
predict

References


See Also


Examples

```r
## First example
data(breast.tumors)
X <- breast.tumors$gene.exp
Y <- breast.tumors$sample$treatment

plsda.breast <- plsda(X, Y, ncomp = 2)
plotIndiv(plsda.breast, ind.names = TRUE, ellipse = TRUE, legend = TRUE)

## Not run:
## Second example
data(liver.toxicity)
X <- liver.toxicity$gene
Y <- liver.toxicity$treatment[, 4]

plsda.liver <- plsda(X, Y, ncomp = 2)
plotIndiv(plsda.liver, ind.names = Y, ellipse = TRUE, legend = TRUE)

## End(Not run)
```

**predict**

Predict Method for (mint). (block). (s)pls(da) methods

**Description**

Predicted values based on PLS models. New responses and variates are predicted using a fitted model and a new matrix of observations.
predict produces predicted values, obtained by evaluating the PLS-derived methods, returned by \( \text{(mint).(block).(s)pls(da)} \) in the frame \( \text{newdata} \). Variates for \( \text{newdata} \) are also returned. Please note that this method performs multilevel decomposition and/or log ratio transformations if needed (\text{multilevel} is an input parameter while \text{logratio} is extracted from \text{object}).

Different prediction distances are proposed for discriminant analysis. The reason is that our supervised models work with a dummy indicator matrix of \( Y \) to indicate the class membership of each sample. The prediction of a new observation results in either a predicted dummy variable (output \text{object$predict}), or a predicted variate (output \text{object$variates}). Therefore, an appropriate distance needs to be applied to those predicted values to assign the predicted class. We propose distances such as ‘maximum distance’ for the predicted dummy variables, ‘Mahalanobis distance’ and ‘Centroids distance’ for the predicted variates.

```
"max.dist" is the simplest method to predict the class of a test sample. For each new individual, the class with the largest predicted dummy variable is the predicted class. This distance performs well in single data set analysis with multiclass problems (PLS-DA).

"centroids.dist" allocates to the new observation the class that minimises the distance between the predicted score and the centroids of the classes calculated on the latent components or variates of the trained model.

"mahalanobis.dist" allocates the new sample the class defined as the centroid distance, but using the Mahalanobis metric in the calculation of the distance.
In practice we found that the centroid-based distances ("centroids.dist" and "mahalanobis.dist"), and specifically the Mahalanobis distance led to more accurate predictions than the maximum distance for complex classification problems and N-integration problems (block.splsda). The centroid distances consider the prediction in dimensional space spanned by the predicted variates, while the maximum distance considers a single point estimate using the predicted scores on the last dimension of the model. The user can assess the different distances, and choose the prediction distance that leads to the best performance of the model, as highlighted from the tune and perf outputs.

More (mathematical) details about the prediction distances are available in the supplemental of the mixOmics manuscript.

For a visualisation of those prediction distances, see background.predict that overlays the prediction area in plotIndiv for a sPLS-DA object.

For MINT objects, the study.test argument is required and provides the grouping factor of newdata.

For multi block analysis (thus block objects), newdata is a list of matrices whose names are a subset of names(object$x) and missing blocks are allowed. Several predictions are returned, either for each block or for all blocks. For non discriminant analysis, the predicted values (predict) are returned for each block and these values are combined by average (AveragedPredict) or weighted average (WeightedPredict), using the weights of the blocks that are calculated as the correlation between a block’s components and the outcome’s components.

For discriminant analysis, the predicted class is returned for each block (class) and each distance (dist) and these predictions are combined by majority vote (MajorityVote) or weighted majority vote (WeightedVote), using the weights of the blocks that are calculated as the correlation between a block’s components and the outcome’s components. NA means that there is no consensus among the block. For PLS-DA and sPLS-DA objects, the prediction area can be visualised in plotIndiv via the background.predict function.

Value

predict produces a list with the following components:

- **predict**: predicted response values. The dimensions correspond to the observations, the response variables and the model dimension, respectively. For a supervised model, it corresponds to the predicted dummy variables.
- **variates**: matrix of predicted variates.
- **B.hat**: matrix of regression coefficients (without the intercept).
- **AveragedPredict**: if more than one block, returns the average predicted values over the blocks (using the predict output)
- **WeightedPredict**: if more than one block, returns the weighted average of the predicted values over the blocks (using the predict and weights outputs)
- **class**: predicted class of newdata for each 1,...,ncomp components.
- **MajorityVote**: if more than one block, returns the majority class over the blocks. NA for a sample means that there is no consensus on the predicted class for this particular sample over the blocks.
predict

WeightedVote  if more than one block, returns the weighted majority class over the blocks. NA for a sample means that there is no consensus on the predicted class for this particular sample over the blocks.

weights  Returns the weights of each block used for the weighted predictions, for each nrepeat and each fold

centroids  matrix of coordinates for centroids.

dist  type of distance requested.

vote  majority vote result for multi block analysis (see details above).

Author(s)

Florian Rohart, Sebastien Dejean, Ignacio Gonzalez, Kim-Anh Le Cao

References


See Also


Examples

data(linnerud)
X <- linnerud$exercise
Y <- linnerud$physiological
linn.pls <- pls(X, Y, ncomp = 2, mode = "classic")

indiv1 <- c(200, 40, 60)
indiv2 <- c(190, 45, 45)
newdata <- rbind(indiv1, indiv2)
colnames(newdata) <- colnames(X)
newdata

default <- predict(linn.pls, newdata)

plotIndiv(linn.pls, comp = 1:2, rep.space = "X-variate", style="graphics", ind.names=FALSE)
points(default$variates[, 1], default$variates[, 2], pch = 19, cex = 1.2)
text(default$variates[, 1], default$variates[, 2],
c("new ind.1", "new ind.2"), pos = 3)

## First example with plsda
data(liver.toxicity)
X <- liver.toxicity$gene
Y <- as.factor(liver.toxicity$treatment[, 4])
predict

```r
## if training is performed on 4/5th of the original data
samp <- sample(1:5, nrow(X), replace = TRUE)
test <- which(samp == 1)  # testing on the first fold
train <- setdiff(1:nrow(X), test)
plsda.train <- plsda(X[train, ], Y[train], ncomp = 2)
test.predict <- predict(plsda.train, X[test, ], dist = "max.dist")
Prediction <- test.predict$class$max.1
cbind(Y = as.character(Y[test]), Prediction)

## Not run:
## Second example with splsda
splsda.train <- splsda(X[train, ], Y[train], ncomp = 2, keepX = c(30, 30))
test.predict <- predict(splsda.train, X[test, ], dist = "max.dist")
Prediction <- test.predict$class$max.1
cbind(Y = as.character(Y[test]), Prediction)

## example with block.splsda=diablo=sgcca and a missing block
data(nutrimouse)
# need to unmap Y for an unsupervised analysis, where Y is included as a data block in data
Y.mat = unmap(nutrimouse$diet)
data = list(gene = nutrimouse$gene, lipid = nutrimouse$lipid, Y = Y.mat)
# with this design, all blocks are connected
design = matrix(c(0,1,1,0,1,1,0), ncol = 3, nrow = 3,
byrow = TRUE, dimnames = list(names(data), names(data)))
# train on 75
ind.train=NULL
for(i in 1:levels(nutrimouse$diet))
  ind.train=c(ind.train,which(nutrimouse$diet==levels(nutrimouse$diet)[i])[1:6])
# training set
gene.train=nutrimouse$gene[ind.train,]
lipid.train=nutrimouse$lipid[ind.train,]
Y.mat.train=Y.mat[ind.train,]
Y.train=nutrimouse$diet[ind.train]
data.train=list(gene=gene.train, lipid=lipid.train, Y=Y.mat.train)
# test set
gene.test=nutrimouse$gene[-ind.train,]
lipid.test=nutrimouse$lipid[-ind.train,]
Y.mat.test=Y.mat[-ind.train,]
Y.test=nutrimouse$diet[-ind.train]
data.test=list(gene=gene.test, lipid=lipid.test)
# example with block.splsda=diablo=sgcca and a missing block
res.train = block.splsda(X=list(gene=gene.train, lipid=lipid.train), Y=Y.train,
ncomp=3,keepX=list(gene=c(10,10,10), lipid=c(5,5,5)))
test.predict = predict(res.train, newdata=data.test[2], method = "max.dist")

## example with mint.splsda
```
data(stemcells)

# training set
ind.test = which(stemcells$study == "3")
gene.train = stemcells$gene[-ind.test,]
Y.train = stemcells$celltype[-ind.test]
study.train = factor(stemcells$study[-ind.test])

# test set
gene.test = stemcells$gene[ind.test,]
Y.test = stemcells$celltype[ind.test]
study.test = factor(stemcells$study[ind.test])

res = mint.splsda(X = gene.train, Y = Y.train, ncomp = 3, keepX = c(10, 5, 15),
study = study.train)
pred = predict(res, newdata = gene.test, study.test = study.test)
data.frame(Truth = Y.test, prediction = pred$class$max.dist)

## End(Not run)

---

**Print Methods for CCA, (s)PLS, PCA and Summary objects**

**Description**

Produce print methods for class "rcc", "pls", "spls", "pca", "rgcca", "sgcca" and "summary".

**Usage**

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

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

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

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

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

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

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

### Arguments

- **x**
  - object of class inheriting from "rcc", "pls", "spls", "pca", "spca", "rgcca", "sgcca" or "summary".
  - not used currently.

### Details

print method for "rcc", "pls", "spls", "pca", "rgcca", "sgcca" class, returns a description of the x object including: the function used, the regularization parameters (if x of class "rcc"), the (s)PLS algorithm used (if x of class "pls" or "spls"), the samples size, the number of variables selected on each of the sPLS components (if x of class "spls") and the available components of the object.

print method for "summary" class, gives the (s)PLS algorithm used (if x of class "pls" or "spls"), the number of variates considered, the canonical correlations (if x of class "rcc"), the number of variables selected on each of the sPLS components (if x of class "spls") and the available components for Communalities Analysis, Redundancy Analysis and Variable Importance in the Projection (VIP).

### Author(s)

Sebastien Dejean, Ignacio Gonzalez and Kim-Anh Le Cao.

### See Also

rcc, pls, spls, vip.

### Examples

```
## print for objects of class 'rcc'
data(nutrimouse)
X <- nutrimouse$lipid
Y <- nutrimouse$gene
nutri.res <- rcc(X, Y, ncomp = 3, lambda1 = 0.064, lambda2 = 0.008)
print(nutri.res)

## print for objects of class 'summary'
more <- summary(nutri.res, cutoff = 0.65)
print(more)

## print for objects of class 'pls'
data(linnerud)
X <- linnerud$exercise
```
rcc <- linnerud$physiological
linn.pls <- pls(X, Y)
print(linn.pls)

## print for objects of class 'spls'
data(liver.toxicity)
X <- liver.toxicity$gene
Y <- liver.toxicity$clinic
toxicity.spls <- spls(X, Y, ncomp = 3, keepX = c(50, 50, 50),
                      keepY = c(10, 10, 10))
print(toxicity.spls)

---

rcc  
Regularized Canonical Correlation Analysis

Description

The function performs the regularized extension of the Canonical Correlation Analysis to seek correlations between two data matrices.

Usage

rcc(X, 
   Y,  
   ncomp = 2,  
   method = "ridge",  
   lambda1 = 0,  
   lambda2 = 0)

Arguments

X  
numeric matrix or data frame \((n \times p)\), the observations on the \(X\) variables. NAs are allowed.

Y  
numeric matrix or data frame \((n \times q)\), the observations on the \(Y\) variables. NAs are allowed.

method  
One of "ridge" or "shrinkage". If "ridge", \(\lambda_1\) and \(\lambda_2\) need to be supplied (see also our function tune.rcc); if "shrinkage", parameters are directly estimated with Strimmer's formula, see below and reference.

ncomp  
the number of components to include in the model. Default to 2.

\(\lambda_1, \lambda_2\)  
a non-negative real. The regularization parameter for the \(X\) and \(Y\) data. Defaults to \(\lambda_1=\lambda_2=0\). Only used if method="ridge"
Details

The main purpose of Canonical Correlations Analysis (CCA) is the exploration of sample correlations between two sets of variables \(X\) and \(Y\) observed on the same individuals (experimental units) whose roles in the analysis are strictly symmetric.

The `cancor` function performs the core of computations but additional tools are required to deal with data sets highly correlated (nearly collinear), data sets with more variables than units by example.

The `rcc` function, the regularized version of CCA, is one way to deal with this problem by including a regularization step in the computations of CCA. Such a regularization in this context was first proposed by Vinod (1976), then developed by Leurgans et al. (1993). It consists in the regularization of the empirical covariances matrices of \(X\) and \(Y\) by adding a multiple of the matrix identity, that is, \(\text{Cov}(X) + \lambda_1 I\) and \(\text{Cov}(Y) + \lambda_2 I\).

When \(\lambda_1=0\) and \(\lambda_2=0\), \(rcc\) performs a classical CCA, if possible (i.e. when \(n > p + q\)).

The shrinkage estimates method = "shrinkage" can be used to bypass `tune.rcc` to choose the shrinkage parameters - which can be long and costly to compute with very large data sets. Note that both functions `tune.rcc` (which uses cross-validation) and the whrinkage parameters (which uses the formula from Schafer and Strimmer) may output different results.

Note: when method = "shrinkage" the input data are centered and scaled for the estimation of the shrinkage parameters and the calculation of the regularised variance-covariance matrices in \(rcc\).

The estimation of the missing values can be performed by the reconstitution of the data matrix using the `nipals` function. Otherwise, missing values are handled by casewise deletion in the `rcc` function.

Value

\(rcc\) returns an object of class "rcc", a list that contains the following components:

- **X**: the original \(X\) data.
- **Y**: the original \(Y\) data.
- **cor**: a vector containing the canonical correlations.
- **lambda**: a vector containing the regularization parameters whether those were input if ridge method or directly estimated with the shrinkage method.
- **loadings**: list containing the estimated coefficients used to calculate the canonical variates in \(X\) and \(Y\).
- **variates**: list containing the canonical variates.
- **names**: list containing the names to be used for individuals and variables.

Author(s)

Sebastien Dejean, Ignacio Gonzalez, Francois Bartolo.

References


See Also

`summary`, `tune_rcc`, `plot.rcc`, `plotIndiv`, `plotVar`, `cim`, `network` and http://www.mixOmics.org for more details.

Examples

```r
## Classic CCA
data(linnerud)
X <- linnerud$exercise
Y <- linnerud$physiological
linn.res <- rcc(X, Y)

## Regularized CCA
data(nutrimouse)
X <- nutrimouse$lipid
Y <- nutrimouse$gene
nutri.res1 <- rcc(X, Y, ncomp = 3, lambda1 = 0.064, lambda2 = 0.008)

## using shrinkage parameters
nutri.res2 <- rcc(X, Y, ncomp = 3, method = 'shrinkage')
nutri.res2$lambda # the shrinkage parameters
```

### selectVar

**Output of selected variables**

**Description**

This function outputs the selected variables on each component for the sparse versions of the approaches (was also generalised to the non sparse versions for our internal functions).
selectVar

Usage

## S3 method for class 'pls'
selectVar(object, comp = 1, block=NULL, ...)

## S3 method for class 'pca'
selectVar(object, comp = 1, block=NULL, ...)

## S3 method for class 'spls'
selectVar(object, comp = 1, block=NULL, ...)

## S3 method for class 'sgcca'
selectVar(object, comp = 1, block=NULL, ...)

## S3 method for class 'rgcca'
selectVar(object, comp = 1, block=NULL, ...)

Arguments

object object of class inheriting from "pls", "spls", "plsda", "splsda", "pca", "spca", "sipca".
comp integer value indicating the component of interest.
block for an object of class "sgcca", the block data sets can be specified as an input vector, for example c(1, 2) for the first two blocks. Default to NULL (all block data sets)
... other arguments.

Details

selectVar provides the variables selected on a given component.

name outputs the name of the selected variables (provided that the input data have colnames) ranked in decreasing order of importance.

value outputs the loading value for each selected variable, the loadings are ranked according to their absolute value.

These functions are only implemented for the sparse versions.

Author(s)

Kim-Anh Le Cao, Florian Rohart.

Examples

data(liver.toxicity)
X = liver.toxicity$gene
Y = liver.toxicity$clinic

# example with sPCA
# ---------------
liver.s pca <- spca(X, ncomp = 1, keepX = 10)
selectVar(liver.s pca, comp = 1)$name
selectVar(liver.s pca, comp = 1)$value
# example with sIPCA
# -------------------
## Not run:
liver.sipca <- sipca(X, ncomp = 3, keepX = rep(10, 3))
selectVar(liver.sipca, comp = 1)

## End(Not run)

# example with sPLS
# -------------------
## Not run:
liver.spls = spls(X, Y, ncomp = 2, keepX = c(20, 40), keepY = c(5, 5))
selectVar(liver.spls, comp = 2)

# example with sPLS-DA
data(srbct)  # an example with no gene name in the data
X = srbct$gene
Y = srbct$class
srbct.splsda = splsda(X, Y, ncomp = 2, keepX = c(5, 10))
select = selectVar(srbct.splsda, comp = 2)
select
# this is a very specific case where a data set has no rownames.
srbct$gene.name[substr(select$select, 2, 5),]

## End(Not run)

# example with sGCCA
# -------------------
## Not run:
data(nutrimouse)

# ! need to unmap the Y factor
Y = unmap(nutrimouse$diet)
data = list(gene = nutrimouse$gene, lipid = nutrimouse$lipid, Y)
# in this design, gene expression and lipids are connected to the diet factor
# and gene expression and lipids are also connected
design = matrix(c(0, 1, 1,
                 1, 0, 1,
                 1, 1, 0), ncol = 3, nrow = 3, byrow = T)
# note: the penalty parameters need to be tuned
wrap.result.sgcca = wrapper.sgcca(X = data, design = design, penalty = c(3, 3, 1),
ncomp = 2,
scheme = "horst", verbose = FALSE)

# variables selected and loadings values on component 1 for the two blocs
selectVar(wrap.result.sgcca, comp = 1, block = c(1, 2))

# variables selected on component 1 for each block
selectVar(wrap.result.sgcca, comp = 1, block = c(1, 2))$'gene'$name
selectVar(wrap.result.sgcca, comp = 1, block = c(1, 2))$'lipid'$name

# variables selected on component 2 for each block
selectVar(wrap.result.sgcca, comp = 2, block = c(1,2))$'gene'$name
selectVar(wrap.result.sgcca, comp = 2, block = c(1,2))$'lipid'$name

# loading value of the variables selected on the first block
selectVar(wrap.result.sgcca, comp = 1, block = 1)$'gene'$value

## End(Not run)

---

### sipca

**Independent Principal Component Analysis**

**Description**

Performs sparse independent principal component analysis on the given data matrix to enable variable selection.

**Usage**

```r
sipca(X, ncomp, mode = c("deflation","parallel"),
       fun = c("logcosh","exp"),
       scale = FALSE, max.iter = 200,
       tol = 1e-04, keepX = rep(50,ncomp),
       w.init=NULL)
```

**Arguments**

- **X**
  a numeric matrix (or data frame) which provides the data for the principal component analysis.

- **ncomp**
  integer, number of independent component to choose. Set by default to 3.

- **mode**
  character string. What type of algorithm to use when estimating the unmixing matrix, (partially) matching one of "deflation", "parallel". Default set to deflation.

- **fun**
  the function used in approximation to neg-entropy in the FastICA algorithm. Default set to logcosh, see details of FastICA.

- **scale**
  a logical value indicating whether rows of the data matrix X should be standardized beforehand.

- **max.iter**
  integer, maximum number of iterations to perform.

- **tol**
  a positive scalar giving the tolerance at which the un-mixing matrix is considered to have converged, see fastICA package.

- **keepX**
  the number of variable to keep on each dimensions.

- **w.init**
  initial un-mixing matrix (unlike FastICA, this matrix is fixed here).
Details

See Details of ipca.
Soft thresholding is implemented on the independent loading vectors to obtain sparse loading vectors and enable variable selection.

Value

pca returns a list with class "ipca" containing the following components:

- `ncomp` the number of principal components used.
- `unmixing` the unmixing matrix of size (ncomp x ncomp)
- `mixing` the mixing matrix of size (ncomp x ncomp x the centered data matrix x the principal components (with sparse independent loadings)
- `loadings` the sparse independent loading vectors
- `kurtosis` the kurtosis measure of the independent loading vectors

Author(s)

Fangzhou Yao and Jeff Coquery.

References


See Also


Examples

data(liver.toxicity)

# implement IPCA on a microarray dataset
sipca.res <- sipca(liver.toxicity$gene, ncomp = 3, mode="deflation", keepX=c(50,50,50))
sipca.res

# samples representation
plotIndiv(sipca.res, ind.names = liver.toxicity$treatment[, 4],
         group = as.numeric(as.factor(liver.toxicity$treatment[, 4])))
## Not run: plotIndiv(sipca.res, cex = 0.01,
         col = as.numeric(as.factor(liver.toxicity$treatment[, 4])),style="3d")
Sparse Principal Components Analysis

Description

Performs a sparse principal components analysis to perform variable selection by using singular value decomposition.

Usage

```r
spca(x, ncomp = 2, center = TRUE, scale = TRUE,
     keepx = rep(ncol(x), ncomp), max.iter = 500,
     tol = 1e-06, logratio = 'none', # one of ('none', 'CLR')
     multilevel = NULL)
```

Arguments

- **X**: a numeric matrix (or data frame) which provides the data for the sparse principal components analysis.
- **ncomp**: integer, the number of components to keep.
- **center**: a logical value indicating whether the variables should be shifted to be zero centered. Alternatively, a vector of length equal the number of columns of \( X \) can be supplied. The value is passed to `scale`.
- **scale**: a logical value indicating whether the variables should be scaled to have unit variance before the analysis takes place. The default is `TRUE`. See details.
- **max.iter**: integer, the maximum number of iterations to check convergence in each component.
- **tol**: a positive real, the tolerance used in the iterative algorithm.
- **keepx**: numeric vector of length `ncomp`, the number of variables to keep in loading vectors. By default all variables are kept in the model. See details.
- **logratio**: one of (`'none'`, `'CLR'`). Specifies the log ratio transformation to deal with compositional values that may arise from specific normalisation in sequencing data. Default to `'none'`.
- **multilevel**: sample information for multilevel decomposition for repeated measurements.
Details

The calculation employs singular value decomposition of the (centered and scaled) data matrix and LASSO to generate sparsity on the loading vectors.

scale = TRUE is highly recommended as it will help obtaining orthogonal sparse loading vectors. keepX is the number of variables to keep in loading vectors. The difference between number of columns of \( X \) and keepX is the degree of sparsity, which refers to the number of zeros in each loading vector.

Note that spca does not apply to the data matrix with missing values. The biplot function for spca is not available.

According to Filzmoser et al., a ILR log ratio transformation is more appropriate for PCA with compositional data. Both CLR and ILR are valid.

Logratio transform and multilevel analysis are performed sequentially as internal pre-processing step, through logratio.transfo and withinVariation respectively.

Logratio can only be applied if the data do not contain any 0 value (for count data, we thus advise the normalise raw data with a 1 offset). For ILR transformation and additional offset might be needed.

Value

spca returns a list with class "spca" containing the following components:

- ncomp: the number of components to keep in the calculation.
- varX: the adjusted cumulative percentage of variances explained.
- keepX: the number of variables kept in each loading vector.
- iter: the number of iterations needed to reach convergence for each component.
- rotation: the matrix containing the sparse loading vectors.
- x: the matrix containing the principal components.

Author(s)

Kim-Anh Le Cao, Fangzhou Yao, Leigh Coonan

References


See Also

pca and http://www.mixOmics.org for more details.
Examples

data(liver.toxicity)
s pca.rat <- spca(liver.toxicity$gene, ncomp = 3, keepX = rep(0, 3))
spca.rat

## variable representation
plotVar(spca.rat, cex = 0.5)
## Not run: plotVar(spca.rat, style="3d")

## samples representation
plotIndiv(spca.rat, ind.names = liver.toxicity$treatment[, 3],
    group = as.numeric(liver.toxicity$treatment[, 3]))
## Not run: plotIndiv(spca.rat, cex = 0.01,
    #col = as.numeric(liver.toxicity$treatment[, 3]), style="3d")
## End(Not run)

# example with multilevel decomposition and CLR log ratio transformation
# ---------------------
# Not run:
data("diverse.16S")
pca.res = pca(X = diverse.16S$data.TSS, ncomp = 5,
    logratio = 'CLR', multilevel = diverse.16S$sample)
plot(pca.res)
plotIndiv(pca.res, ind.names = FALSE, group = diverse.16S$bodysite, title = '16S diverse data',
    legend=TRUE)
## End(Not run)

spls

Sparse Partial Least Squares (sPLS)

Description

Function to perform sparse Partial Least Squares (sPLS). The sPLS approach combines both integration and variable selection simultaneously on two data sets in a one-step strategy.

Usage

spls(X,
Y,
ncomp = 2,
mode = c("regression", "canonical", "invariant", "classic"),
keepX.constraint=NULL,
keepY.constraint=NULL,
keepX,
keepY,
scale = TRUE,
tol = 1e-06,
max.iter = 100,
near.zero.var = FALSE,
logratio="none",
multilevel=NULL)

Arguments

X numeric matrix of predictors. NAs are allowed.
Y numeric vector or matrix of responses (for multi-response models). NAs are allowed. For multilevel analysis, a data frame of up to two columns is accepted.
ncomp the number of components to include in the model (see Details). Default is set to from one to the rank of X.
mode character string. What type of algorithm to use, (partially) matching one of "regression", "canonical", "invariant" or "classic". See Details.
keepX.constraint A list containing which variables of X are to be kept on each of the first PLS-components
keepY.constraint A list containing which variables of Y are to be kept on each of the first PLS-components
keepX numeric vector of length ncomp, the number of variables to keep in X-loadings. By default all variables are kept in the model.
keepY numeric vector of length ncomp, the number of variables to keep in Y-loadings. By default all variables are kept in the model.
scale boolean. If scale = TRUE, each block is standardized to zero means and unit variances (default: TRUE)
tol Convergence stopping value.
max.iter integer, the maximum number of iterations.
near.zero.var boolean, see the internal nearZeroVar function (should be set to TRUE in particular for data with many zero values). Setting this argument to FALSE (when appropriate) will speed up the computations. Default value is FALSE
logratio one of ('none','CLR'). Default to 'none'
multilevel Design matrix for repeated measurement analysis, where multilevel decomposition is required. For a one factor decomposition, the repeated measures on each individual, i.e. the individuals ID is input as the first column. For a 2 level factor decomposition then 2nd AND 3rd columns indicate those factors. See examples.

Details

spls function fit sPLS models with 1,...,ncomp components. Multi-response models are fully supported. The X and Y datasets can contain missing values.

The type of algorithm to use is specified with the mode argument. Four PLS algorithms are available: PLS regression ("regression"), PLS canonical analysis ("canonical"), redundancy analysis ("invariant") and the classical PLS algorithm ("classic") (see References and ?pls for more details).
The estimation of the missing values can be performed by the reconstitution of the data matrix using the nipals function. Otherwise, missing values are handled by casewise deletion in the spls function without having to delete the rows with missing data.

logratio transform and multilevel analysis are performed sequentially as internal pre-processing step, through logratio.transfo and withinVariation respectively.

Multilevel sPLS enables the integration of data measured on two different data sets on the same individuals. This approach differs from multilevel sPLS-DA as the aim is to select subsets of variables from both data sets that are highly positively or negatively correlated across samples. The approach is unsupervised, i.e. no prior knowledge about the sample groups is included.

Value

spls returns an object of class "spls", a list that contains the following components:

- **X**: the centered and standardized original predictor matrix.
- **Y**: the centered and standardized original response vector or matrix.
- **ncomp**: the number of components included in the model.
- **mode**: the algorithm used to fit the model.
- **keepX**: number of X variables kept in the model on each component.
- **keepY**: number of Y variables kept in the model on each component.
- **keepX.constraint**: if a constraint model was chosen, the list of X variables that were constraint to be selected on each of the sPLS components.
- **keepY.constraint**: if a constraint model was chosen, the list of Y variables that were constraint to be selected on each of the sPLS components.
- **variates**: list containing the variates.
- **loadings**: list containing the estimated loadings for the X and Y variates.
- **names**: list containing the names to be used for individuals and variables.
- **tol**: the tolerance used in the iterative algorithm, used for subsequent S3 methods.
- **iter**: Number of iterations of the algorithm for each component.
- **max.iter**: the maximum number of iterations, used for subsequent S3 methods.
- **nzv**: list containing the zero- or near-zero predictors information.
- **scale**: whether scaling was applied per predictor.
- **logratio**: whether log ratio transformation for relative proportion data was applied, and if so, which type of transformation.
- **explained.variance**: amount of variance explained per component (note that contrary to PCA, this amount may not decrease as the aim of the method is not to maximise the variance, but the covariance between data sets).
- **input.X**: numeric matrix of predictors in X that was input, before any saling / logratio / multilevel transformation.
- **mat.c**: matrix of coefficients from the regression of X / residual matrices X on the X-variates, to be used internally by predict.
- **defl.matrix**: residual matrices X for each dimension.
Author(s)
Sebastien Dejean, Ignacio Gonzalez and Kim-Anh Le Cao.

References
Sparse PLS: canonical and regression modes:
On multilevel analysis:

See Also
*pls*, *summary*, *plotIndiv*, *plotVar*, *cim*, *network*, *predict*, *perf* and [http://www.mixOmics.org](http://www.mixOmics.org) for more details.

Examples
```r
data(liver.toxicity)
X <- liver.toxicity$gene
Y <- liver.toxicity$clinic

toxicity.spls <- spls(X, Y, ncomp = 2, keepX = c(50, 50),
keepY = c(10, 10))
```

```
## Second example: one-factor multilevel analysis with sPLS, selecting a subset of variables
#-----------------------------------------------
## Not run:
data(liver.toxicity)
# note: we made up those data, pretending they are repeated measurements
repeat.indiv <- c(1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 3, 3, 4, 3, 4, 3, 4, 5, 6, 5, 5, 6, 7, 7, 8, 6, 7, 8, 7, 8, 8, 9, 10, 9, 10, 11, 9, 9,
```
summary(as.factor(repeat.indiv)) # 16 rats, 4 measurements each

# this is a spls (unsupervised analysis) so no need to mention any factor in design
# we only perform a one level variation split
design <- data.frame(sample = repeat.indiv)
res.spls.1level <- spls(X = liver.toxicity$gene,
Y = liver.toxicity$clinic,
multilevel = design,
ncomp = 3,
keepX = c(50, 50, 50), keepY = c(5, 5, 5),
mode = 'canonical')

# set up colors and pch for plotIndiv
col.stimu <- 1:nlevels(design$stimu)
plotIndiv(res.spls.1level, rep.space = 'X-variate', ind.names = FALSE,
group = liver.toxicity$treatment$dose.Group,
pch = 20, main = 'Gene expression subspace',
legend = TRUE)

plotIndiv(res.spls.1level, rep.space = 'Y-variate', ind.names = FALSE,
group = liver.toxicity$treatment$dose.Group,
pch = 20, main = 'Clinical measurements subspace',
legend = TRUE)

plotIndiv(res.spls.1level, rep.space = 'XY-variate', ind.names = FALSE,
group = liver.toxicity$treatment$dose.Group,
pch = 20, main = 'Both Gene expression and Clinical subspaces',
legend = TRUE)

## End(Not run)

## Third example: two-factor multilevel analysis with sPLS, selecting a subset of variables
#---------------------------------------------------------------
## Not run:
data(liver.toxicity)
dose <- as.factor(liver.toxicity$treatment$dose.Group)
time <- as.factor(liver.toxicity$treatment$time.Group)
# note: we made up those data, pretending they are repeated measurements
repeat.indiv <- c(1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 3, 3, 4, 4, 3, 3, 4, 4, 5, 6, 5, 5,
6, 5, 6, 7, 7, 8, 6, 7, 8, 7, 8, 9, 10, 9, 10, 11, 9, 9,
10, 11, 12, 12, 10, 11, 12, 11, 12, 13, 14, 13, 14, 13, 14,
13, 14, 15, 16, 15, 16, 15, 16, 16)
summary(as.factor(repeat.indiv)) # 16 rats, 4 measurements each
design <- data.frame(sample = repeat.indiv, dose = dose, time = time)

res.spls.2level = spls(liver.toxicity$gene,
Y = liver.toxicity$clinic,
multilevel = design,
ncomp=2,
keepX = c(10,10), keepY = c(5,5))

## End(Not run)

### Sparse Partial Least Squares Discriminant Analysis (sPLS-DA)

**Description**

Function to perform sparse Partial Least Squares to classify samples (supervised analysis) and select variables.

**Usage**

```r
splsda(X,  
Y,  
ncomp = 2,  
mode = c("regression", "canonical", "invariant", "classic"),  
keepX,  
keepX.constraint=NULL,  
scale = TRUE,  
tol = 1e-06,  
max.iter = 100,  
near.zero.var = FALSE,  
logratio="none", # one of "none", "CLR"  
multilevel=NULL)
```

**Arguments**

- **X**: numeric matrix of predictors. NAs are allowed.
- **Y**: a factor or a class vector for the discrete outcome.
- **ncomp**: the number of components to include in the model (see Details). Default is set to from one to the rank of X.
- **mode**: character string. What type of algorithm to use, (partially) matching one of "regression", "canonical", "invariant" or "classic". See Details.
- **keepX.constraint**: A list containing which variables of X are to be kept on each of the first PLS-components.
- **keepX**: numeric vector of length ncomp, the number of variables to keep in X-loadings. By default all variables are kept in the model.
- **scale**: boolean. If scale = TRUE, each block is standardized to zero means and unit variances (default: TRUE)
- **tol**: Convergence stopping value.
max.iter integer, the maximum number of iterations.

near.zero.var boolean, see the internal nearZeroVar function (should be set to TRUE in particular for data with many zero values). Setting this argument to FALSE (when appropriate) will speed up the computations. Default value is FALSE.

logratio one of ('none','CLR') specifies the log ratio transformation to deal with compositional values that may arise from specific normalisation in sequencing data. Default to 'none'.

multilevel sample information for multilevel decomposition for repeated measurements. A numeric matrix or data frame indicating the repeated measures on each individual, i.e. the individuals ID. See examples.

Details
splsda function fits an sPLS model with 1,...,ncomp components to the factor or class vector y. The appropriate indicator (dummy) matrix is created. Logratio transform and multilevel analysis are performed sequentially as internal pre-processing step, through logratio.transfo and withinVariation respectively.

Logratio can only be applied if the data do not contain any 0 value (for count data, we thus advise the normalise raw data with a 1 offset).

More details about the PLS modes in ?pls.

Value
splsda returns an object of class "splsda", a list that contains the following components:

X the centered and standardized original predictor matrix.
Y the centered and standardized indicator response vector or matrix.
ind.mat the indicator matrix.
ncomp the number of components included in the model.
keepX number of X variables kept in the model on each component.
variates list containing the variates.
loadings list containing the estimated loadings for the X and Y variates.
names list containing the names to be used for individuals and variables.
nzv list containing the zero- or near-zero predictors information.
tol the tolerance used in the iterative algorithm, used for subsequent S3 methods
iter Number of iterations of the algorithm for each component
max.iter the maximum number of iterations, used for subsequent S3 methods
scale boolean indicating whether the data were scaled in MINT S3 methods
logratio whether logratio transformations were used for compositional data
explained_variance amount of variance explained per component (note that contrary to PCA, this amount may not decrease as the aim of the method is not to maximise the variance, but the covariance between X and the dummy matrix Y).
mat.c matrix of coefficients from the regression of X / residual matrices X on the X-variates, to be used internally by predict.
defl.matrix residual matrices X for each dimension.
Author(s)
Florian Rohart, Ignacio Gonzalez, Kim-Anh Le Cao.

References

See Also

Examples
```r
## First example
data(breast.tumors)
X <- breast.tumors$gene.exp
# Y will be transformed as a factor in the function,
# but we set it as a factor to set up the colors.
Y <- as.factor(breast.tumors$sample$treatment)
res <- splsda(X, Y, ncomp = 2, keepX = c(25, 25))

# individual names appear
plotindiv(res, ind.names = Y, legend = TRUE, ellipse = TRUE)

## Second example: one-factor analysis with sPLS-DA, selecting a subset of variables
# as in the paper Liquet et al.
#---------------------------------------------
data(vac18)
X <- vac18$genes
Y <- vac18$stimulation
# sample indicates the repeated measurements
design <- data.frame(sample = vac18$sample)
Y = data.frame(stimul = vac18$stimulation)

# multilevel sPLS-DA model
res.l1level <- splsda(X, Y = Y, ncomp = 3, multilevel = design,
keepX = c(30, 137, 123))
```
# set up colors for plotIndiv
col.stim <- c("darkblue", "purple", "green4", "red3")
plotIndiv(res.1level, ind.names = Y, col.per.group = col.stim)

## Third example: two-factor analysis with sPLS-DA, selecting a subset of variables
# as in the paper Liquet et al.
#---------------------------------------------------------------
## Not run:
data(vac18.simulated)  # simulated data

X <- vac18.simulated$genes
design <- data.frame(sample = vac18.simulated$sample)
Y = data.frame( stimu = vac18.simulated$stimulation,
                time = vac18.simulated$time)

res.2level <- splsda(X, Y = Y, ncomp = 2, multilevel = design,
                      keepX = c(200, 200))

plotIndiv(res.2level, group = Y$stimu, ind.names = vac18.simulated$time,
          legend = TRUE, style = 'lattice')

## End(Not run)

## Fourth example: with more than two classes
#---------------------------------------------------------------
## Not run:
data(liver.toxicity)
X <- as.matrix(liver.toxicity$gene)
# Y will be transformed as a factor in the function,
# but we set it as a factor to set up the colors.
Y <- as.factor(liver.toxicity$treatment[, 4])

splsda.liver <- splsda(X, Y, ncomp = 2, keepX = c(20, 20))

# individual name is set to the treatment
plotIndiv(splsda.liver, ind.names = Y, ellipse = TRUE, legend = TRUE)

## End(Not run)

## Fifth example: 16S data with multilevel decomposition and log ratio transformation
#---------------------------------------------------------------
## Not run:
splsda.16S = splsda(
X = diverse.16S$data.TSS,  # TSS normalised data
Y = diverse.16S$bodysite,
multilevel = diverse.16S$sample, # multilevel decomposition
ncomp = 2,
keepX = c(10, 150),
logratio= 'CLR') # CLR log ratio transformation

plotIndiv(splsda.16S, ind.names = FALSE, pch = 16, ellipse = TRUE, legend = TRUE)
# srbct

Small version of the small round blue cell tumors of childhood data

**Description**

This data set from Khan *et al.*, (2001) gives the expression measure of 2308 genes measured on 63 samples.

**Usage**

```r
data(srbct)
```

**Format**

A list containing the following components:

- **gene**  data frame with 63 rows and 2308 columns. The expression measure of 2308 genes for the 63 subjects.
- **class**  A class vector containing the class tumour of each case (4 classes in total).
- **gene.name**  data frame with 2308 rows and 2 columns containing further information on the genes.

**Source**

http://research.nhgri.nih.gov/microarray/Supplement

**References**

Human Stem Cells Data

Description

This data set contains the expression of a random subset of 400 genes in 125 samples from 4 independent studies and 3 cell types.

Usage

data(stemcells)

Format

A list containing the following components:

- **gene**: data matrix with 125 rows and 400 columns. Each row represents an experimental sample, and each column a single gene.
- **celltype**: a factor indicating the cell type of each sample.
- **study**: a factor indicating the study from which the sample was extracted.

Details

This data set contains the expression of a random subset of 400 genes in 125 samples from 4 independent studies and 3 cell types. Those studies can be combined and analysed using the MINT procedure.

References


study_split

divides a data matrix in a list of matrices defined by a factor

Description

study_split divides a data matrix in a list of matrices defined by a study input.

Usage

study_split(data, study)
Arguments

- data: numeric matrix of predictors
- study: grouping factor indicating which samples are from the same study

Value

study_split simply returns a list of the same length as the number of levels of study that contains submatrices of data.

Author(s)

Florian Rohart

See Also

mint.pls, mint.spls, mint.plsda, mint.splsda.

Examples

```r
data = stemcells$gene
ex = stemcells$study
data.list = study_split(data, exp)
names(data.list)
lapply(data.list, dim)
table(exp)
```

---

**Summary Methods for CCA and PLS objects**

**Description**

Produce summary methods for class "rcc", "pls" and "spls".

**Usage**

```r
## S3 method for class 'rcc'
summary(object, what = c("all", "communalities", "redundancy"),
cutoff = NULL, digits = 4, ...)

## S3 method for class 'pls'
summary(object, what = c("all", "communalities", "redundancy",
"VIP"), digits = 4, keep.var = FALSE, ...)

## S3 method for class 'spls'
summary(object, what = c("all", "communalities", "redundancy",
"VIP"), digits = 4, keep.var = FALSE, ...)
```
Arguments

object  object of class inheriting from "rcc", "pls" or "spls".
cutoff real between 0 and 1. Variables with all correlations components below this
cutoff in absolute value are not showed (see Details).
digits integer, the number of significant digits to use when printing. Defaults to 4.
what character string or vector. Should be a subset of c("all", "summarised", "communalities", "redundancy", "VIP"). "VIP" is only available for (s)PLS. See Details.
keep.var boolean. If TRUE only the variables with loadings not zero (as selected by spls) are showed. Defaults to FALSE.

Details

The information in the rcc, pls or spls object is summarised, it includes: the dimensions of X and Y data, the number of variates considered, the canonical correlations (if object of class "rcc") and the (s)PLS algorithm used (if object of class "pls" or "spls") and the number of variables selected on each of the sPLS components (if x of class "spls").

"communalities" in what gives Communalities Analysis. "redundancy" display Redundancy Analysis. "VIP" gives the Variable Importance in the Projection (VIP) coefficients fit by pls or spls. If what is "all", all are given.

For class "rcc", when a value to cutoff is specified, the correlations between each variable and the equiangular vector between X- and Y-variates are computed. Variables with at least one correlation componente bigger than cutoff are showed. The defaults is cutoff=NULL all the variables are given.

Value

The function summary returns a list with components:

ncomp the number of components in the model.
cor the canonical correlations.
cutoff the cutoff used.
keep.var list containing the name of the variables selected.
mode the algorithm used in pls or spls.
Cm list containing the communalities.
Rd list containing the redundancy.
VIP matrix of VIP coefficients.
what subset of c("all", "communalities", "redundancy", "VIP").
digits the number of significant digits to use when printing.
method method used: rcc, pls or spls.
Author(s)
Sebastien Dejean, Ignacio Gonzalez and Kim-Anh Le Cao.

See Also
rcc, pls, spls, vip.

Examples

```r
## summary for objects of class 'rcc'
data(nutrimouse)
X <- nutrimouse$lipid
Y <- nutrimouse$gene
nutri.res <- rcc(X, Y, ncomp = 3, lambda1 = 0.064, lambda2 = 0.008)
more <- summary(nutri.res, cutoff = 0.65)

## summary for objects of class 'pls'
data(linnerud)
X <- linnerud$exercise
Y <- linnerud$physiological
linn.pls <- pls(X, Y)
more <- summary(linn.pls)

## summary for objects of class 'spls'
data(liver.toxicity)
X <- liver.toxicity$gene
Y <- liver.toxicity$clinic
toxicity.spls <- spls(X, Y, ncomp = 3, keepX = c(50, 50, 50),
                      keepY = c(10, 10, 10))
more <- summary(toxicity.spls, what = "redundancy", keep.var = TRUE)
```

tune

Generic function to choose the parameters in the different methods in mixOmics

Description
Wrapper of all tuning functions.

Usage

tune(method, X, Y, multilevel, ncomp, study, # mint.splslsda
test.keepX = c(5, 10, 15), # all but pca, rcc
test.keepY = NULL, # rcc, multilevel
already.tested.X, # all but pca, rcc
already.tested.Y, # multilevel
mode = "regression", # multilevel
nrepeat = 1, # multilevel, splsda
grid1 = seq(0.001, 1, length = 5), # rcc
grid2 = seq(0.001, 1, length = 5), # rcc
validation = "Mfold", # all but pca
folds = 10, # all but pca
dist = "max.dist", # all but pca, rcc
measure = c("BER"), # all but pca, rcc
auc = FALSE,
progressBar = TRUE, # all but pca, rcc
near.zero.var = FALSE, # all but pca, rcc
logratio = "none", # all but pca, rcc
center = TRUE, # pca
scale = TRUE, # mint, splsda
max.iter = 100, # pca
tol = 1e-09,
light.output = TRUE # mint, splsda
)

Arguments

method
This parameter is used to pass all other argument to the suitable function. method has to be one of the following: "spls", "splsda", "mint.splsda", "rcc", "pca".

X numeric matrix of predictors. NA are allowed.

Y Either a factor or a class vector for the discrete outcome, or a numeric vector or matrix of continuous responses (for multi-response models).

multilevel Design matrix for multilevel analysis (for repeated measurements) that indicates the repeated measures on each individual, i.e. the individuals ID. See Details.

ncomp the number of components to include in the model.

study grouping factor indicating which samples are from the same study.

test.keepX numeric vector for the different number of variables to test from the X data set.

test.keepY If method = 'spls', numeric vector for the different number of variables to test from the Y data set.

already.tested.X Optional, if ncomp > 1 A numeric vector indicating the number of variables to select from the X data set on the firsts components.

already.tested.Y if method = 'spls' and if(ncomp > 1) numeric vector indicating the number of variables to select from the Y data set on the first components.

mode character string. What type of algorithm to use, (partially) matching one of "regression", "canonical", "invariant" or "classic". See Details.

nrepeat Number of times the Cross-Validation process is repeated.
grid1, grid2  vector numeric defining the values of lambda1 and lambda2 at which cross-validation score should be computed. Defaults to grid1=grid2=seq(0.001, 1, length=5).

validation  character. What kind of (internal) validation to use, matching one of "Mfold" or "loo" (see below). Default is "Mfold".

testNkeepX  the folds in the Mfold cross-validation. See Details.

dist  distance metric to use for splsda to estimate the classification error rate, should be a subset of "centroids.dist", "mahalanobis.dist" or "max.dist" (see Details).

measure  Two misclassification measure are available: overall misclassification error overall or the Balanced Error Rate BER

auc  if TRUE calculate the Area Under the Curve (AUC) performance of the model.

progressBar  by default set to TRUE to output the progress bar of the computation.

near.zero.var  boolean, see the internal nearZeroVar function (should be set to TRUE in particular for data with many zero values). Default value is FALSE

logratio  one of ('none', 'CLR'). Default to 'none'

center  a logical value indicating whether the variables should be shifted to be zero centered. Alternately, a vector of length equal the number of columns of \(X\) can be supplied. The value is passed to scale.

scale  a logical value indicating whether the variables should be scaled to have unit variance before the analysis takes place. The default is FALSE for consistency with prcomp function, but in general scaling is advisable. Alternatively, a vector of length equal the number of columns of \(X\) can be supplied. The value is passed to scale.

max.iter  integer, the maximum number of iterations for the NIPALS algorithm.

tol  a positive real, the tolerance used for the NIPALS algorithm.

light.output  if set to FALSE, the prediction/classification of each sample for each of test, keepX and each comp is returned.

Details

For more details, please see the help file corresponding to your method, e.g. tune.splsda. Note that only the arguments used in the tune function corresponding to method are passed on.

Some details on the use of the nrepeat argument are provided in ?perf.


Value

Depending on the type of analysis performed and the input arguments, a list that may contain:

error.rate  returns the prediction error for each test, keepX on each component, averaged across all repeats and subsampling folds. Standard deviation is also output. All error rates are also available as a list.

choice.keepX  returns the number of variables selected (optimal keepX) on each component.
choice.ncomp  For supervised models; returns the optimal number of components for the model for each prediction distance using one-sided t-tests that test for a significant difference in the mean error rate (gain in prediction) when components are added to the model. See more details in Rohart et al 2016 Suppl. For more than one block, an optimal ncomp is returned for each prediction framework.

error.rate.class  returns the error rate for each level of \(Y\) and for each component computed with the optimal keepX

predict  Prediction values for each sample, each test.keepX, each comp and each repeat. Only if light.output=FALSE

class  Predicted class for each sample, each test.keepX, each comp and each repeat. Only if light.output=FALSE

auc  AUC mean and standard deviation if the number of categories in \(Y\) is greater than 2, see details above. Only if auc = TRUE

cor.value  only if multilevel analysis with 2 factors: correlation between latent variables.

Author(s)
Florian Rohart

References

DIABLO:

mixOmics main publication:

MINT:


Chavent, Marie and Patouille, Brigitte (2003). Calcul des coefficients de regression et du PRESS en regression PLS1. Modulad n, 30 1-11. (this is the formula we use to calculate the Q2 in perf.pls and perf.spls)


sparse PLS regression mode:
One-sided t-tests (suppl material):


See Also

Examples

```r
## sPLS-DA
## Not run:
data(breast.tumors)
X <- breast.tumors$gene.exp
Y <- as.factor(breast.tumors$sample$treatment)
tune <- tune(method = "splsda", X, Y, ncomp=1, nrepeat=10, logratio="none",
           test.keepX = c(5, 10, 15), folds=10, dist="max.dist", progressBar = TRUE)
plot(tune)

## End(Not run)

## mint.splsda
## Not run:
data(stemcells)
data.stemcells$gene
type.id = stemcells$celltype
exp = stemcells$study

out = tune(method="mint.splsda", X=data,Y=type.id, ncomp=2, study=exp, test.keepX=seq(1,10,1))
out$choice.keepX
plot(out)

## End(Not run)
```

---

tune.block.splsda  Tuning function for block.splsda method (N-integration with sparse Discriminant Analysis)

---
Description

Computes M-fold or Leave-One-Out Cross-Validation scores based on a user-input grid to determine the optimal parsity parameters values for method block.splsda.

Usage

tune.block.splsda(x, y, indY, ncomp = 2, test.keepX, already.tested.X, validation = "Mfold", folds = 10, dist = "max.dist", measure = "BER", weighted = TRUE, progressBar = TRUE, max.iter = 100, near.zero.var = FALSE, nrepeat = 1, design, scheme, mode, scale = TRUE, bias, init, tol = 1e-06, verbose, light.output = TRUE, cpus, name.save=NULL)

Arguments

X numeric matrix of predictors. NAs are allowed.
Y if(method = 'spls') numeric vector or matrix of continuous responses (for multi-response models) NAs are allowed.
indY To be supplied if Y is missing, indicates the position of the matrix / vector response in the list X
ncomp the number of components to include in the model.
test.keepX A list of length the number of blocks in X (without the outcome). Each entry of this list is a numeric vector for the different keepX values to test for that specific block.
already.tested.X Optional, if ncomp > 1 A numeric vector indicating the number of variables to select from the X data set on the firsts components.
validation character. What kind of (internal) validation to use, matching one of "Mfold" or "loo" (see below). Default is "Mfold".

folds the folds in the Mfold cross-validation. See Details.

dist distance metric to use for splsda to estimate the classification error rate, should be a subset of "centroids.dist", "mahalanobis.dist" or "max.dist" (see Details).

measure Two misclassification measure are available: overall misclassification error overall or the Balanced Error Rate BER.

weighted tune using either the performance of the Majority vote or the Weighted vote.

progressBar by default set to TRUE to output the progress bar of the computation.

max.iter integer, the maximum number of iterations.

near.zero.var boolean, see the internal nearZeroVar function (should be set to TRUE in particular for data with many zero values). Default value is FALSE

nrepeat Number of times the Cross-Validation process is repeated.

design numeric matrix of size (number of blocks in X) x (number of blocks in X) with 0 or 1 values. A value of 1 (0) indicates a relationship (no relationship) between the blocks to be modelled. If Y is provided instead of indY, the design matrix is changed to include relationships to Y.

scheme Either "horst", "factorial" or "centroid". Default = centroid, see reference.

mode character string. What type of algorithm to use, (partially) matching one of "regression", "canonical", "invariant" or "classic". Default = regression.

scale boolean. If scale = TRUE, each block is standardized to zero means and unit variances. Default = TRUE.

bias boolean. A logical value for biaised or unbiaised estimator of the var/cov. Default = FALSE.

init Mode of initialization use in the algorithm, either by Singular Value Decomposition of the product of each block of X with Y ("svd") or each block independently ("svd.single"). Default = svd.

tol Convergence stopping value.

verbose if set to TRUE, reports progress on computing.

light.output if set to FALSE, the prediction/classification of each sample for each of test, keepX and each comp is returned.

cpus Number of cpus to use when running the code in parallel.

name.save character string for the name of the file to be saved.

Details

This tuning function should be used to tune the keepX parameters in the block.splsda function (N-integration with sparse Discriminant Analysis).

M-fold or LOO cross-validation is performed with stratified subsampling where all classes are represented in each fold.
If `validation = "Mfold", M-fold cross-validation is performed. The number of folds to generate is to be specified in the argument `folds`.

If `validation = "loo", leave-one-out cross-validation is performed. By default `folds` is set to the number of unique individuals.

All combination of `test.keepX` values are tested. A message informs how many will be fitted on each component for a given `test.keepX`.


**Value**

A list that contains:

- `error.rate` returns the prediction error for each `test.keepX` on each component, averaged across all repeats and subsampling folds. Standard deviation is also output. All error rates are also available as a list.

- `choice.keepX` returns the number of variables selected (optimal `keepX`) on each component, for each block.

- `choice.ncomp` returns the optimal number of components for the model fitted with `choice.keepX`.

- `error.rate.class` returns the error rate for each level of `Y` and for each component computed with the optimal `keepX`.

- `predict` Prediction values for each sample, each `test.keepX`, each comp and each repeat. Only if `light.output=FALSE`

- `class` Predicted class for each sample, each `test.keepX`, each comp and each repeat. Only if `light.output=FALSE`

- `cor.value` compute the correlation between latent variables for two-factor sPLS-DA analysis.

**Author(s)**


**References**

Method:

mixOmics manuscript:

**See Also**

Examples

```r
## Not run:
data("breast.TCGA")
# this is the X data as a list of mRNA and miRNA; the Y data set is a single data set of proteins
data = list(mrna = breast.TCGA$data.train$mrna, mirna = breast.TCGA$data.train$mirna, protein = breast.TCGA$data.train$protein)
# set up a full design where every block is connected
# could also consider other weights, see our mixOmics manuscript
design = matrix(1, ncol = length(data), nrow = length(data),
               dimnames = list(names(data), names(data))
diag(diag) = 0

design
# set number of component per data set
ncomp = 5

# Tuning the first two components
# ---------------

# definition of the keepX value to be tested for each block mRNA miRNA and protein
# names of test.keepX must match the names of 'data'
test.keepX = list(mrna = seq(10,40,20), mirna = seq(10,30,10), protein = seq(1,10,5))

# the following may take some time to run, note that for through tuning
# nrepeat should be > 1
tune = tune.block.splsda(X = data, Y = breast.TCGA$data.train$subtype, ncomp = ncomp, test.keepX = test.keepX, design = design, nrepeat = 3)
tune$choice.ncomp

tune$choice.keepX

# Only tuning the second component
# ---------------

already.mrna = 4 # 4 variables selected on comp1 for mrna
already.mirna = 2 # 2 variables selected on comp1 for mirna
already.prot = 1 # 1 variables selected on comp1 for protein

already.tested.X = list(mrna = already.mrna, mirna = already.mirna, prot = already.prot)

tune = tune.block.splsda(X = data, Y = breast.TCGA$data.train$subtype, ncomp = 2, test.keepX = test.keepX, design = design, already.tested.X = already.tested.X)
tune$choice.keepX

## End(Not run)
```
tune.mint.splsda

Estimate the parameters of mint.splsda method

Description
Computes Leave-One-Group-Out-Cross-Validation (LOGOCV) scores on a user-input grid to determine optimal values for the sparsity parameters in mint.splsda.

Usage

tune.mint.splsda(X, Y, ncomp = 1, study, test.keepX = c(5, 10, 15), already.tested.X, dist = "max.dist", measure = "BER", auc = FALSE, progressBar = TRUE, scale = TRUE, tol = 1e-06, max.iter = 100, near.zero.var = FALSE, light.output = TRUE )

Arguments

X numeric matrix of predictors. NAs are allowed.
Y Outcome. Numeric vector or matrix of responses (for multi-response models)
ncomp Number of components to include in the model (see Details). Default to 1
study grouping factor indicating which samples are from the same study
test.keepX numeric vector for the different number of variables to test from the X data set
already.tested.X if ncomp > 1 Numeric vector indicating the number of variables to select from the X data set on the firsts components
dist only applies to an object inheriting from "plsda" or "splsda" to evaluate the classification performance of the model. Should be a subset of "max.dist", "centroids.dist", "mahalanobis.dist". Default is "all". See predict.
measure Two misclassification measure are available: overall misclassification error overall or the Balanced Error Rate BER
auc if TRUE calculate the Area Under the Curve (AUC) performance of the model.
progressBar by default set to TRUE to output the progress bar of the computation.
scale boolean. If scale = TRUE, each block is standardized to zero means and unit variances (default: TRUE)
tol Convergence stopping value.
max.iter integer, the maximum number of iterations.
near.zero.var boolean, see the internal nearZeroVar function (should be set to TRUE in particular for data with many zero values). Default value is FALSE
light.output if set to FALSE, the prediction/classification of each sample for each of test.keepX and each comp is returned.
Details

This function performs a Leave-One-Group-Out-Cross-Validation (LOGOCV), where each of study is left out once. It returns a list of variables of X that were selected on each of the ncomp components. Then, a mint.splsda can be performed with keepX set as the output choice.keepX.

All component 1 : ncomp are tuned, except the first ones for which a already.tested.X is provided. See examples below.

The function outputs the optimal number of components that achieve the best performance based on the overall error rate or BER. The assessment is data-driven and similar to the process detailed in (Rohart et al., 2016), where one-sided t-tests assess whether there is a gain in performance when adding a component to the model. Our experience has shown that in most case, the optimal number of components is the number of categories in Y - 1.

More details about the prediction distances in ?predict and the referred publication.

Value

The returned value is a list with components:

- error.rate: returns the prediction error for each test.keepX on each component, averaged across all repeats and subsampling folds. Standard deviation is also output. All error rates are also available as a list.
- choice.keepX: returns the number of variables selected (optimal keepX) on each component.
- choice.ncomp: returns the optimal number of components for the model fitted with $choice.keepX
- error.rate.class: returns the error rate for each level of Y and for each component computed with the optimal keepX.
- predict: Prediction values for each sample, each test.keepX and each comp.
- class: Predicted class for each sample, each test.keepX and each comp.

Author(s)

Florian Rohart

References


mixOmics manuscript:


See Also

**tune.multilevel**

**Examples**

```r
data(stemcells)
data = stemcells$gene
type.id = stemcells$celltype
exp = stemcells$study

res = mint.splsda(X=data,Y=type.id,ncomp=3,keepX=c(10,5,15),study=exp)
out = tune.mint.splsda(X=data,Y=type.id,ncomp=2,near.zero.var=FALSE,
                      study=exp,test.keepX=seq(1,10,1))

out$choice.ncomp
out$choice.keepX

## Not run:
out = tune.mint.splsda(X=data,Y=type.id,ncomp=2,near.zero.var=FALSE,
                      study=exp,test.keepX=seq(1,10,1))
out$choice.keepX

## only tune component 2 and keeping 10 genes on comp1
out = tune.mint.splsda(X=data,Y=type.id,ncomp=2, study=exp,
                      already.tested.X = c(10),
                      test.keepX=seq(1,10,1))
out$choice.keepX
# gives the number of variables selected (which comes from out$choice.keepX.constraint)

## End(Not run)
```

---

**tune.multilevel**

*Tuning functions for multilevel analyses*

**Description**

These functions were implemented to help tuning the variable selection parameters in the multilevel analyses.

**Usage**

```r
tune.multilevel(X,
    Y,
    multilevel ,
    ncomp = 1,
    test.keepX = c(5, 10, 15),
    test.keepY = NULL,
    already.tested.X = NULL,
    already.tested.Y = NULL,
    method,
    mode = "regression",
```
validation = "Mfold",
folds = 10,
dist = "max.dist",
measure = "BER",
auc = FALSE,
progressBar = TRUE,
near.zero.var = FALSE,
logratio = "none",
nrepeat=1,
light.output = TRUE)

Arguments

X numeric matrix of predictors. NAs are allowed.
Y if(method = 'spls') numeric vector or matrix of continuous responses (for multi-response models) NAs are allowed.
multilevel Design matrix for multilevel analysis (for repeated measurements). A numeric matrix or data frame. For a one level factor decomposition, the input is a vector indicating the repeated measures on each individual, i.e. the individuals ID. For a two level decomposition with splsda models, the two factors are included in Y. Finally for a two level decomposition with spls models, 2nd AND 3rd columns in design indicate those factors (see example in ?splsda and ?spls).
ncomp the number of components to include in the model.
test.keepX numeric vector for the different number of variables to test from the X data set
test.keepY If method = 'spls', numeric vector for the different number of variables to test from the Y data set
already.tested.X if ncomp > 1 numeric vector indicating the number of variables to select rom the X data set on the pfirst components
already.tested.Y if method = 'spls' and if(ncomp > 1) numeric vector indicating the number of variables to select from the Y data set on the first components
method character string. Which multivariate method and type of analysis to choose, matching one of 'splsda' (Discriminant Analysis) or 'spls' (unsupervised integrative analysis). See Details.
mode character string. What type of algorithm to use, (partially) matching one of "regression", "canonical", "invariant" or "classic". See Details.
validation character. What kind of (internal) validation to use, matching one of "Mfold" or "loo" (see below). Default is "Mfold".
folds the folds in the Mfold cross-validation. See Details.
dist distance metric to use for splsda to estimate the classification error rate, should be a subset of "centroids.dist", "mahalanobis.dist" or "max.dist" (see Details).
measure Two misclassification measure are available: overall misclassification error overall or the Balanced Error Rate BER
This tuning function should be used to tune the parameters when using a variance decomposition ('multilevel') with a repeated measurement design, see also details in ?predict.splsda.

If method = 'splsda', a distance metric must be used, see help(predict.splsda) for details about the distances.

The function outputs the optimal number of components that achieve the best performance based on the overall error rate or BER. The assessment is data-driven and similar to the process detailed in (Rohart et al., 2016), where one-sided t-tests assess whether there is a gain in performance when adding a component to the model. Our experience has shown that in most case, the optimal number of components is the number of categories in Y - 1.

For a sPLS-DA one-factor analysis, M-fold cross-validation is performed, internally the training data is decomposed into within-subject variation.

For a sPLS-DA two-factor analysis, the correlation between components from the within-subject variation of X and a matrix including the two factors (design[,,-1]) is computed on the whole data set. We cannot obtain a cross-validation error rate as for the spls-DA one-factor analysis because of the difficulty to decompose and predict the within matrices within each fold.

For a sPLS two-factor analysis a sPLS canonical mode is run, and the correlation between components from the within-subject variation of X and Y is computed on the whole data set.

If validation = "Mfold", M-fold cross-validation is performed. How many folds to generate is selected by specifying the number of folds in folds. The folds also can be supplied as a list of vectors containing the indexes defining each fold as produced by split.

If validation = "1oo", leave-one-out cross-validation is performed. By default folds is set to the number of unique individuals.

If auc = TRUE and there are more than 2 categories in Y, the Area Under the Curve is averaged using one-vs-all comparison. Note however that the AUC criteria may not be particularly insightful as the prediction threshold we use in sPLS-DA differs from an AUC threshold (sPLS-DA relies on prediction distances for predictions, see details in ?predict.splsda).


Value

Depending on the type of analysis performed, a list that may contain:
error.rate returns the prediction error for each test.keepX on each component, averaged across all repeats and subsampling folds. Standard deviation is also output. All error rates are also available as a list.

choice.keepX returns the number of variables selected (optimal keepX) on each component.

choice.ncomp returns the optimal number of components in the sPLS-DA model.

error.rate.class returns the error rate for each level of Y and for each component computed with the optimal keepX.

predict Prediction values for each sample, each test.keepX, each comp and each repeat. Only if light.output=FALSE

class Predicted class for each sample, each test.keepX, each comp and each repeat. Only if light.output=FALSE

auc AUC mean and standard deviation if the number of categories in Y is greater than 2, see details above. Only if auc = TRUE

cor.value only if multilevel analysis with 2 factors: correlation between latent variables.

Author(s)
Benoit Liquet, Kim-Anh Le Cao, Florian Rohart.

References
On multilevel analysis:


mixOmics manuscript:

See Also

Examples
## First example: one-factor analysis with sPLS-DA
## Not run:
data(vac18.simulated) # simulated data
design <- data.frame(sample = vac18.simulated$sample)

result.ex1 = tune.multilevel(X = vac18.simulated$genes,
                           Y = vac18.simulated$stimulation,
                           multilevel = design,
                           ncomp=2,
test.keepX=c(5, 10, 15),
already.tested.X = c(50),
method = 'splsda',
dist = 'mahalanobis.dist',
validation = 'loo')

# overall error rate
result.ex1$error.rate
# classification error rate per class after 2 components
result.ex1$error.rate.class

## Second example: two-factor analysis with sPLS-DA
## Not run:
data(liver.toxicity)
dose <- as.factor(liver.toxicity$treatment$Dose.Group)
time <- as.factor(liver.toxicity$treatment$Time.Group)
# note: we made up those data, pretending they are repeated measurements
repeat.indiv <- c(1, 2, 1, 2, 1, 2, 1, 2, 3, 3, 4, 3, 4, 3, 4, 4, 5, 6, 5, 5,
6, 5, 6, 7, 7, 8, 6, 7, 8, 7, 8, 9, 10, 9, 10, 11, 11, 9, 9,
10, 11, 12, 12, 10, 11, 11, 12, 13, 14, 13, 14, 13, 14,
13, 14, 15, 16, 15, 16, 15, 16, 15, 16)
summary(as.factor(repeat.indiv)) # 16 rats, 4 measurements each
design <- data.frame(sample = repeat.indiv)

result.ex2 = tune.multilevel(liver.toxicity$gene,
    Y = data.frame(dose, time),
    multilevel = design,
    ncomp=2,
    test.keepX=c(5, 10, 15),
    already.tested.X = c(50),
    method = 'splsda',
    dist = 'mahalanobis.dist')

result.ex2

## End(Not run)

## Third example: one-factor integrative analysis with sPLS
## Not run:
data(liver.toxicity)
# note: we made up those data, pretending they are repeated measurements
repeat.indiv <- c(1, 2, 1, 2, 1, 2, 1, 2, 3, 3, 4, 3, 4, 3, 4, 4, 5, 6, 5, 5,
6, 5, 6, 7, 7, 8, 6, 7, 8, 7, 8, 9, 10, 9, 10, 11, 11, 9, 9,
10, 11, 12, 12, 10, 11, 12, 13, 14, 13, 14, 13, 14,
13, 14, 15, 16, 15, 16, 15, 16, 15, 16)
summary(as.factor(repeat.indiv)) # 16 rats, 4 measurements each

# here we are only interested in a one level variation split since spls is an unsupervised method
design <- data.frame(sample = repeat.indiv)
result.ex3 = tune.multilevel(X = liver.toxicity$gene, Y = liver.toxicity$clinic, 
multilevel = design, 
mode = 'canonical', 
ncomp=2, 
test.keepX=c(5, 10, 15), 
test.keepY=c(2,3), 
already.tested.X = c(50), already.tested.Y = c(5), 
method = 'spls')

result.ex3

## End(Not run)

tune.pca

Tune the number of principal components in PCA

Description

tune.pca can be used to quickly visualise the proportion of explained variance for a large number of principal components in PCA.

Usage

tune.pca(X, ncomp = NULL, center = TRUE, scale = FALSE, 
max.iter = 500, tol = 1e-09, logratio = 'none', 
V = NULL, multilevel = NULL)

Arguments

X a numeric matrix (or data frame) which provides the data for the principal components analysis. It can contain missing values.
ncomp integer, the number of components to initially analyse in tune.pca to choose a final ncomp for pca. If NULL, function sets ncomp = min(nrow(X), ncol(X))
center a logical value indicating whether the variables should be shifted to be zero centered. Alternately, a vector of length equal the number of columns of X can be supplied. The value is passed to scale.
scale a logical value indicating whether the variables should be scaled to have unit variance before the analysis takes place. The default is FALSE for consistency with prcomp function, but in general scaling is advisable. Alternatively, a vector of length equal the number of columns of X can be supplied. The value is passed to scale.
max.iter integer, the maximum number of iterations for the NIPALS algorithm.
tol a positive real, the tolerance used for the NIPALS algorithm.
logratio one of ('none','CLR','ILR'). Default to 'none'
Matrix used in the logratio transformation id provided.

multilevel Design matrix for multilevel analysis (for repeated measurements).

Details

The calculation is done either by a singular value decomposition of the (possibly centered and
scaled) data matrix, if the data is complete or by using the NIPALS algorithm if there is data missing.
Unlike princomp, the print method for these objects prints the results in a nice format and the plot
method produces a bar plot of the percentage of variance explaned by the principal components
(PCs).

When using NIPALS (missing values), we make the assumption that the first (min(ncol(X),
nrow(X)) principal components will account for 100 % of the explained variance.

Note that scale= TRUE cannot be used if there are zero or constant (for center = TRUE) variables.
Components are omitted if their standard deviations are less than or equal to comp.toi times the
standard deviation of the first component. With the default null setting, no components are omitted.
Other settings for comp.toi could be comp.toi = sqrt(Machine$double.eps), which would
omit essentially constant components, or comp.toi = 0.

logratio transform and multilevel analysis are performed sequentially as internal pre-processing
step, through logratio.transfo and withinVariation respectively.

Value

tune.pca returns a list with class "tune.pca" containing the following components:

sdev the square root of the eigenvalues of the covariance/correlation matrix, though
the calculation is actually done with the singular values of the data matrix).

explained.variance the proportion of explained variance accounted for by each principal component
is calculated using the eigenvalues

cum.var the cumulative proportion of explained variance accounted for by the sequential
accumulation of principal components is calculated using the sum of the
proportion of explained variance

Author(s)

Ignacio Gonzalez and Leigh Coonan

See Also

nipals, biplot, plotIndiv, plotVar and http://www.mixOmics.org for more details.

Examples

data(liver.toxicity)
tune <- tune.pca(liver.toxicity$gene, center = TRUE, scale = TRUE)
tune
Estimate the parameters of regularization for Regularized CCA

Description

Computes leave-one-out or M-fold cross-validation scores on a two-dimensional grid to determine optimal values for the parameters of regularization in rcc.

Usage

tune.rcc(X, Y, grid1 = seq(0.001, 1, length = 5),
grid2 = seq(0.001, 1, length = 5),
validation = c("loo", "Mfold"),
folds = 10, plot = TRUE)

Arguments

X numeric matrix or data frame \((n \times p)\), the observations on the \(X\) variables. NAs are allowed.

Y numeric matrix or data frame \((n \times q)\), the observations on the \(Y\) variables. NAs are allowed.

grid1, grid2 vector numeric defining the values of \(\lambda_1\) and \(\lambda_2\) at which cross-validation score should be computed. Defaults to grid1=grid2=seq(0.001, 1, length=5).

validation character string. What kind of (internal) cross-validation method to use, (partially) matching one of "loo" (leave-one-out) or "Mfold" (M-folds). See Details.

folds positive integer. Number of folds to use if validation="Mfold". Defaults to folds=10.

plot logical argument indicating whether a image map should be plotted by calling the imgcv function.

Details

If validation="Mfold", M-fold cross-validation is performed by calling Mfold. When folds is given, the elements of folds should be integer vectors specifying the indices of the validation sample and the argument \(M\) is ignored. Otherwise, the folds are generated. The number of cross-validation folds is specified with the argument \(M\).

If validation="loo", leave-one-out cross-validation is performed by calling the loo function. In this case the arguments folds and \(M\) are ignored.

The estimation of the missing values can be performed by the reconstitution of the data matrix using the nipals function. Otherwise, missing values are handled by casewise deletion in the rcc function.
Value

The returned value is a list with components:

- `opt.lambda1`, `opt.lambda2`: value of the parameters of regularization on which the cross-validation method reached it optimal.
- `opt.score`: the optimal cross-validation score reached on the grid.
- `grid1`, `grid2`: original vectors `grid1` and `grid2`.
- `mat`: matrix containing the cross-validation score computed on the grid.

Author(s)

Sebastien Dejean and Ignacio Gonzalez.

See Also


Examples

data(nutrimouse)
X <- nutrimouse$lipid
Y <- nutrimouse$gene

## this can take some seconds
## Not run:
tune.rcc(X, Y, validation = "Mfold")

## End(Not run)

tune.splsda tuning functions for sPLS-DA method

Description

Computes M-fold or Leave-One-Out Cross-Validation scores on a user-input grid to determine optimal values for the sparsity parameters in splsda.

Usage

tune.splsda(X, Y, ncomp = 1, test.keepX = c(5, 10, 15), already.tested.X, validation = "Mfold", folds = 10, dist = "max.dist", measure = "BER", auc = FALSE, progressBar = TRUE, max.iter = 100, near.zero.var = FALSE, nrepeat = 1, logratio = c('none','CLR'), multilevel = NULL, light.output = TRUE, cpus)
Arguments

- **X**: numeric matrix of predictors. NAs are allowed.
- **Y**: if `method = 'spls'` numeric vector or matrix of continuous responses (for multi-response models) NAs are allowed.
- **ncomp**: the number of components to include in the model.
- **test.keepX**: numeric vector for the different number of variables to test from the \( X \) data set already tested. \( X \)
  Optional, if ncomp > 1 A numeric vector indicating the number of variables to select from the \( X \) data set on the firsts components.
- **validation**: character. What kind of (internal) validation to use, matching one of “Mfold” or "loo" (see below). Default is "Mfold".
- **folds**: the folds in the Mfold cross-validation. See Details.
- **dist**: distance metric to use for splsda to estimate the classification error rate, should be a subset of "centroids.dist", "mahalanobis.dist" or "max.dist" (see Details).
- **measure**: Two misclassification measure are available: overall misclassification error overall or the Balanced Error Rate BER
- **auc**: if TRUE calculate the Area Under the Curve (AUC) performance of the model.
- **progressBar**: by default set to TRUE to output the progress bar of the computation.
- **max.iter**: integer, the maximum number of iterations.
- **near.zero.var**: boolean, see the internal `nearZeroVar` function (should be set to TRUE in particular for data with many zero values). Default value is FALSE
- **nrepeat**: Number of times the Cross-Validation process is repeated.
- **logratio**: one of ('none','CLR'). Default to 'none'
- **multilevel**: Design matrix for multilevel analysis (for repeated measurements) that indicates the repeated measures on each individual, i.e. the individuals ID. See Details.
- **light.output**: if set to FALSE, the prediction/classification of each sample for each of test.keepX and each comp is returned.
- **cpus**: Number of cpus to use when running the code in parallel.

Details

This tuning function should be used to tune the parameters in the splsda function (number of components and number of variables in `keepX` to select).

For a sPLS-DA, M-fold or LOO cross-validation is performed with stratified subsampling where all classes are represented in each fold.

If `validation = "loo"`, leave-one-out cross-validation is performed. By default `folds` is set to the number of unique individuals.

The function outputs the optimal number of components that achieve the best performance based on the overall error rate or BER. The assessment is data-driven and similar to the process detailed in (Rohart et al., 2016), where one-sided t-tests assess whether there is a gain in performance when
adding a component to the model. Our experience has shown that in most case, the optimal number of components is the number of categories in \( Y - 1 \).

For sPLS-DA multilevel one-factor analysis, M-fold or LOO cross-validation is performed where all repeated measurements of one sample are in the same fold. Note that logratio transform and the multilevel analysis are performed internally and independently on the training and test set.

For a sPLS-DA multilevel two-factor analysis, the correlation between components from the within-subject variation of \( X \) and the cond matrix is computed on the whole data set. The reason why we cannot obtain a cross-validation error rate as for the spls-DA one-factor analysis is because of the difficulty to decompose and predict the within matrices within each fold.

For a sPLS two-factor analysis a sPLS canonical mode is run, and the correlation between components from the within-subject variation of \( X \) and \( Y \) is computed on the whole data set.

If validation = "Mfold", M-fold cross-validation is performed. How many folds to generate is selected by specifying the number of folds in folds.

If auc = TRUE and there are more than 2 categories in \( Y \), the Area Under the Curve is averaged using one-vs-all comparison. Note however that the AUC criteria may not be particularly insightful as the prediction threshold we use in sPLS-DA differs from an AUC threshold (sPLS-DA relies on prediction distances for predictions, see ?predict.splsda for more details).

More details about the prediction distances in ?predict and the referred publication.

Value

Depending on the type of analysis performed, a list that contains:

- **error.rate** returns the prediction error for each test.keepX on each component, averaged across all repeats and subsampling folds. Standard deviation is also output. All error rates are also available as a list.

- **choice.keepX** returns the number of variables selected (optimal keepX) on each component.

- **choice.ncomp** returns the optimal number of components for the model fitted with $choice.keepX

- **error.rate.class** returns the error rate for each level of \( Y \) and for each component computed with the optimal keepX

- **predict** Prediction values for each sample, each test.keepX, each comp and each repeat. Only if light.output=FALSE

- **class** Predicted class for each sample, each test.keepX, each comp and each repeat. Only if light.output=FALSE

- **auc** AUC mean and standard deviation if the number of categories in \( Y \) is greater than 2, see details above. Only if auc = TRUE

- **cor.value** only if multilevel analysis with 2 factors: correlation between latent variables.

Author(s)

Kim-Anh Le Cao, Benoit Gautier, Francois Bartolo, Florian Rohart.
References

mixOmics manuscript:

See Also


Examples

## First example: analysis with sPLS-DA
## Not run:
data(breast.tumors)
X = breast.tumors$gene.exp
Y = as.factor(breast.tumors$sample$treatment)
tune = tune.splsda(X, Y, ncomp = 1, nrepeat = 10, logratio = "none",
  test.keepX = c(5, 10, 15), folds = 10, dist = "max.dist",
  progressBar = TRUE)

# 5 components, optimising 'keepX' and 'ncomp'
tune = tune.splsda(X, Y, ncomp = 5, test.keepX = c(5, 10, 15),
  folds = 10, dist = "max.dist", nrepeat = 5, progressBar = TRUE)

tune$choice.ncomp
tune$choice.keepX
plot(tune)

## End(Not run)

## only tune component 3 and 4
## keeping 5 and 10 variables on the first two components respectively
## Not run:
tune = tune.splsda(X = X, Y = Y, ncomp = 4,
  already.tested.X = c(5, 10),
  test.keepX = seq(1,10,2), progressBar = TRUE)

## End(Not run)

## Second example: multilevel one-factor analysis with sPLS-DA
## Not run:
data(vac18)
X = vac18$genes
Y = vac18$stimulation
# sample indicates the repeated measurements
design = data.frame(sample = vac18$sample)

tune = tune.splsda(X, Y = Y, ncomp = 3, nrepeat = 10, logratio = "none",

## End(Not run)
unmap

## Dummy matrix for an outcome factor

### Description

Converts a class or group vector or factor into a matrix of indicator variables.

### Usage

```r
unmap(classification, groups=NULL, noise=NULL)
```

### Arguments

- **classification**: A numeric or character vector or factor. Typically the distinct entries of this vector would represent a classification of observations in a data set.
- **groups**: A numeric or character vector indicating the groups from which `classification` is drawn. If not supplied, the default is to assumed to be the unique entries of `classification`.
- **noise**: A single numeric or character value used to indicate the value of `groups` corresponding to noise.

### Value

An $n$ by $K$ matrix of $(0,1)$ indicator variables, where $n$ is the length of samples and $K$ the number of classes in the outcome.

If a `noise` value of symbol is designated, the corresponding indicator variables are relocated to the last column of the matrix.

Note: - you can remap an unmap vector using the function `map` from the package `mclust`. - this function should be used to unmap an outcome vector as in the non-supervised methods of mixOmics. For other supervised analyses such as (s)PLS-DA, (s)gccDA this function is used internally.

### References


Examples

data(nutrimouse)
Y = unmap(nutrimouse$diet)
Y
data = list(gene = nutrimouse$gene, lipid = nutrimouse$lipid, Y = Y)
# data could then used as an input in wrapper.rgcca, which is not, technically,
# a supervised method, see wrapper.rgcca

Description

The data come from a trial evaluating a vaccine based on HIV-1 lipopeptides in HIV-negative volunteers. The vaccine (HIV-1 LIPO-5 ANRS vaccine) contains five HIV-1 amino acid sequences coding for Gag, Pol and Nef proteins. This data set contains the expression measure of a subset of 1000 genes from purified in vitro stimulated Peripheral Blood Mononuclear Cells from 42 repeated samples (12 unique vaccinated participants) 14 weeks after vaccination, 6 hours after in vitro stimulation by either (1) all the peptides included in the vaccine (LIPO-5), or (2) the Gag peptides included in the vaccine (GAG+) or (3) the Gag peptides not included in the vaccine (GAG-) or (4) without any stimulation (NS).

Usage

data(vac18)

Format

A list containing the following components:

gene data frame with 42 rows and 1000 columns. The expression measure of 1000 genes for the 42 samples (PBMC cells from 12 unique subjects).
stimulation is a factor of 42 elements indicating the type of in vitro simulation for each sample.
sample is a vector of 42 elements indicating the unique subjects (for example the value '1' correspond to the first patient PBMC cells). Note that the design of this study is unbalanced.
tab.prob.gene is a data frame with 1000 rows and 2 columns, indicating the Illumina probe ID and the gene name of the annotated genes.

Details

This is a subset of the original study for illustrative purposes.

References

Simulated data based on the vac18 study for multilevel analysis

Description

Simulated data based on the vac18 study to illustrate the use of the multilevel analysis for one and two-factor analysis with sPLS-DA. This data set contains the expression simulated of 500 genes.

Usage

data(vac18.simulated)

Format

A list containing the following components:

- **genes**: data frame with 48 rows and 500 columns. The simulated expression of 500 genes for 48 subjects.
- **sample**: a vector indicating the repeated measurements on each unique subject. See Details.
- **stimulation**: a factor indicating the stimulation condition on each sample.
- **time**: a factor indicating the time condition on each sample.

Details

In this cross-over design, repeated measurements are performed 12 experiments units (or unique subjects) for each of the 4 stimulations.

The simulation study was based on a mixed effects model (see reference for details). Ten clusters of 100 genes were generated. Amongst those, 4 clusters of genes discriminate the 4 stimulations (denoted LIPO5, GAG+, GAG- and NS) as follows:

- 2 gene clusters discriminate (LIPO5, GAG+) versus (GAG-, NS)
- 2 gene clusters discriminate LIPO5 versus GAG+, while GAG+ and NS have the same effect
- 2 gene clusters discriminate GAG- versus NS, while LIPO5 and GAG+ have the same effect
- the 4 remaining clusters represent noisy signal (no stimulation effect)

Only a subset of those genes are presented here (to save memory space).

References

vip  

Variable Importance in the Projection (VIP)

Description
The function vip computes the influence on the Y-responses of every predictor X in the model.

Usage
vip(object)

Arguments
object object of class inheriting from "pls", "plsdha", "spls" or "splsda".

Details
Variable importance in projection (VIP) coefficients reflect the relative importance of each X variable for each X variate in the prediction model. VIP coefficients thus represent the importance of each X variable in fitting both the X- and Y-variates, since the Y-variates are predicted from the X-variates.

VIP allows to classify the X-variables according to their explanatory power of Y. Predictors with large VIP, larger than 1, are the most relevant for explaining Y.

Value
vip produces a matrix of VIP coefficients for each X variable (rows) on each variate component (columns).

Author(s)
Sebastien Dejean and Ignacio Gonzalez.

References

See Also
pls, spls, summary.
withinVariation

Examples

```r
data(linnerud)
X <- linnerud$exercise
Y <- linnerud$physiological
linn.pls <- pls(X, Y)

linn.vip <- vip(linn.pls)

barplot(linn.vip,
        beside = TRUE, col = c("lightblue", "mistyrose", "lightcyan"),
        ylim = c(0, 1.7), legend = rownames(linn.vip),
        main = "Variable importance in the Projection", font.main = 4)
```

withinVariation  Within matrix decomposition for repeated measurements (cross-over design)

Description

This function is internally called by pca, pls, spls, plsda and splsda functions for cross-over design data, but can be called independently prior to any kind of multivariate analyses.

Usage

```r
withinVariation(X, design)
```

Arguments

- `X` numeric matrix of predictors. NAs are allowed.
- `design` a numeric matrix or data frame. The first column indicates the repeated measures on each individual, i.e. the individuals ID. The 2nd and 3rd columns are to split the variation for a 2 level factor.

Details

withinVariation function decomposes the Within variation in the X data set. The resulting $X_w$ matrix is then input in the multilevel function.

One or two-factor analyses are available.

Value

withinVariation simply returns the $X_w$ within matrix, which can be input in the other multivariate approaches already implemented in mixOmics (i.e. spls or splsda, see multilevel, but also pca or ipca).

Author(s)

Benoit Liquet, Kim-Anh Le Cao, Benoit Gautier, Ignacio Gonzalez.
References

On multilevel analysis:

See Also

`spls`, `splsd`, `plotIndiv`, `plotVar`, `mixOmics`, `network`.

Examples

```r
## Example: one-factor analysis matrix decomposition
#--------------------------------------------------------
data(vac18)
X <- vac18$genes
# in design we only need to mention the repeated measurements to split the one level variation
design <- data.frame(sample = vac18$sample)

Xw <- withinVariation(X = X, design = design)
# multilevel PCA
res.pca.1level <- pca(Xw, ncomp = 3)

# compare a normal PCA with a multilevel PCA for repeated measurements.
# note: PCA makes the assumptions that all samples are independent,
# so this analysis is flawed and you should use a multilevel PCA instead
res.pca <- pca(X, ncomp = 3)

# set up colors for plotIndiv
col.stim <- c("darkblue", "purple", "green4", "red3")
col.stim <- col.stim[as.numeric(vac18$stimulation)]

# plotIndiv comparing both PCA and PCA multilevel
plotIndiv(res.pca, ind.names = vac18$stimulation, group = col.stim)
title(main = 'PCA ')
plotIndiv(res.pca.1level, ind.names = vac18$stimulation, group = col.stim)
title(main = 'PCA multilevel')
```

wrapper.rgcca

**mixOmics wrapper for Regularised Generalised Canonical Correlation Analysis (rgcca)**

Description

Wrapper function to perform Regularized Generalised Canonical Correlation Analysis (rGCCA), a generalised approach for the integration of multiple datasets. For more details, see the `help(rgcca)` from the **RGCCA** package.
Usage

wrapper.rgccca(X,
    design = 1 - diag(length(X)),
    tau = rep(1, length(X)),
    ncomp = 1,
    keepX.constraint, keepX,
    scheme = "horst",
    scale = TRUE,
    bias = FALSE,
    init = "svd.single",
    tol = .Machine$double.eps,
    verbose = FALSE,
    max.iter=1000,
    near.zero.var = FALSE)

Arguments

X a list of data sets (called 'blocks') matching on the same samples. Data in the list should be arranged in samples x variables. NAs are not allowed.

design numeric matrix of size (number of blocks in X) x (number of blocks in X) with values between 0 and 1. Each value indicates the strength of the relationship to be modelled between two blocks using sGCCA; a value of 0 indicates no relationship, 1 is the maximum value. If Y is provided instead of indY, the design matrix is changed to include relationships to Y.

tau numeric vector of length the number of blocks in X. Each regularization parameter will be applied on each block and takes the value between 0 (no regularization) and 1. If tau = "optimal" the shrinkage parameters are estimated for each block and each dimension using the Schafer and Strimmer (2005) analytical formula.

ncomp the number of components to include in the model. Default to 1.

keepX.constraint A list of same length as X. Each entry keepX.constraint[[i]] is a list containing which variables of X[[i]] are to be kept on each of the first PLS-components

keepX A vector of same length as X. Each entry keepX[i] is the number of X[[i]]-variables kept in the model on the last components (once all keepX.constraint[[i]] are used).

scheme Either "horst", "factorial" or "centroid" (Default: "horst").

scale boolean. If scale = TRUE, each block is standardized to zero means and unit variances (default: TRUE)

bias boolean. A logical value for biased or unbiased estimator of the var/cov (defaults to FALSE).

init Mode of initialization use in the algorithm, either by Singular Value Decomposition of the product of each block of X with Y ("svd") or each block independently ("svd.single"). Default to "svd.single".
tol          Convergence stopping value.
verbose      if set to TRUE, reports progress on computing.
max.iter     integer, the maximum number of iterations.
near.zero.var boolean, see the internal nearZeroVar function (should be set to TRUE in particular for data with many zero values). Setting this argument to FALSE (when appropriate) will speed up the computations. Default value is FALSE.

Details

This wrapper function performs rGCCA (see RGCCA) with $1, \ldots, n_{\text{comp}}$ components on each block data set. A supervised or unsupervised model can be run. For a supervised model, the unmap function should be used as an input data set. More details can be found on the package RGCCA.

Value

wrapper.rgcca returns an object of class "rgcca", a list that contains the following components:

data         the input data set (as a list).
design       the input design.
variates     the sgcca components.
loadings     the loadings for each block data set (outer weight vector).
loadings.star the loadings, standardised.
tau          the input tau parameter.
scheme       the input scheme.
ncomp        the number of components included in the model for each block.
crit         the convergence criterion.
AVE          Indicators of model quality based on the Average Variance Explained (AVE): AVE(for one block), AVE(outer model), AVE(inner model). 
names        list containing the names to be used for individuals and variables.

More details can be found in the references.

Author(s)


References


See Also

wrapper.rgcca, plotIndiv, plotVar, wrapper.sgcca and http://www.mixOmics.org for more details.
Examples

```r
data(nutrimouse)
# need to unmap the Y factor diet
Y = unmap(nutrimouse$diet)
data = list(gene = nutrimouse$gene, lipid = nutrimouse$lipid, Y = Y)
# with this design, gene expression and lipids are connected to the diet factor
# design = matrix(c(0,0,1, 0,0,1, 1,1,0), ncol = 3, nrow = 3, byrow = TRUE)
# with this design, gene expression and lipids are connected to the diet factor
# and gene expression and lipids are also connected
design = matrix(c(0,1,1, 1,0,1, 1,1,0), ncol = 3, nrow = 3, byrow = TRUE)
#note: the tau parameter is the regularization parameter
wrap.result.rgcca = wrapper.rgcca(X = data, design = design, tau = c(1, 1, 0),
ncomp = 2,
scheme = "centroid", verbose = FALSE)
```

Description

Wrapper function to perform Sparse Generalised Canonical Correlation Analysis (sGCCA), a generalised approach for the integration of multiple datasets. For more details, see the `help(sgcca)` from the `RGCCA` package.

Usage

```r
wrapper.sgcca(X, 
design = 1 - diag(length(X)), 
penalty = NULL, 
ncomp = 1, 
keepX.constraint, 
keepX, 
scheme = "horst", 
mode="canonical", 
scale = TRUE, 
bias = TRUE, 
init = "svd.single", 
tol = .Machine$double.eps, 
verbose = FALSE, 
max.iter=1000, 
near.zero.var = FALSE)
```
Arguments

X  a list of data sets (called 'blocks') matching on the same samples. Data in the
list should be arranged in samples x variables. NAs are not allowed.

design numeric matrix of size (number of blocks in X) x (number of blocks in X) with
values between 0 and 1. Each value indicates the strength of the relationship
to be modelled between two blocks using sGCCA; a value of 0 indicates no
relationship, 1 is the maximum value. If Y is provided instead of indY, the
design matrix is changed to include relationships to Y.

penalty numeric vector of length the number of blocks in X. Each penalty parameter will
be applied on each block and takes the value between 0 (no variable selected)
and 1 (all variables included).

ncomp the number of components to include in the model. Default to 1.

keepX.constraint A list of same length as X. Each entry keepX.constraint[[i]] is a list containing
which variables of X[[i]] are to be kept on each of the first PLS-components

keepX A vector of same length as X. Each entry keepX[i] is the number of X[[i]]-
variables kept in the model on the last components (once all keepX.constraint[[i]]
are used).

scheme Either "horst", "factorial" or "centroid" (Default: "horst").

mode character string. What type of algorithm to use, (partially) matching one of
"regression", "canonical", "invariant" or "classic". See Details.

scale boolean. If scale = TRUE, each block is standardized to zero means and unit
variances (default: TRUE)

bias boolean. A logical value for biased or unbiased estimator of the var/cov (de-
defaults to FALSE).

init Mode of initialization use in the algorithm, either by Singular Value Decompos-
tion of the product of each block of X with Y ("svd") or each block independ-
tently ("svd.single"). Default to "svd.single".

tol Convergence stopping value.

verbose if set to TRUE, reports progress on computing.

max.iter integer, the maximum number of iterations.

near.zero.var boolean, see the internal nearZeroVar function (should be set to TRUE in par-
ticular for data with many zero values). Setting this argument to FALSE (when
appropriate) will speed up the computations. Default value is FALSE

Details

This wrapper function performs sGCCA (see RGCCA) with 1,...,ncomp components on each
block data set. A supervised or unsupervised model can be run. For a supervised model, the unmap
function should be used as an input data set. More details can be found on the package RGCCA.

Note that this function is the same as block.spls with different default arguments.

More details about the PLS modes in ?pls.
Value

wrapper.sgcca returns an object of class "sgcca", a list that contains the following components:

data the input data set (as a list).
design the input design.
variates the sgcca components.
loadings the loadings for each block data set (outer wieght vector).
loadings.star the loadings, standardised.
penalty the input penalty parameter.
scheme the input schme.
ncomp the number of components included in the model for each block.
crit the convergence criterion.
AVE Indicators of model quality based on the Average Variance Explained (AVE): AVE(for one block), AVE(outer model), AVE(inner model)..
names list containing the names to be used for individuals and variables.

More details can be found in the references.

Author(s)


References


See Also

wrapper.sgcca, plotIndiv, plotVar, wrapper.rgcca and http://www.mixOomics.org for more details.

Examples

```r
## Not run:
data(nutrimouse)
# need to unmap the Y factor diet if you pretend this is not a classification pb.
# see also the function block.spIsda for discriminant analysis where you dont
# need to unmap Y.
Y = unmap(nutrimouse$diet)
data = list(gene = nutrimouse$gene, lipid = nutrimouse$lipid, Y = Y)
# with this design, gene expression and lipids are connected to the diet factor
# design = matrix(c(0,0,1,
# 0,0,1,
# 1,1,0), ncol = 3, nrow = 3, byrow = TRUE)
```
# with this design, gene expression and lipids are connected to the diet factor
# and gene expression and lipids are also connected
design = matrix(c(0, 1, 1,
                  1, 0, 1,
                  1, 1, 0), ncol = 3, nrow = 3, byrow = TRUE)

#note: the penalty parameters will need to be tuned
wrap.result.sgcca = wrapper.sgcca(x = data, design = design, penalty = c(3, 5, 1),
                                  ncomp = 2,
                                  scheme = "centroid", verbose = FALSE)

wrap.result.sgcca
#did the algo converge?
wrap.result.sgcca$crit  # yes

## End(Not run)

---

**yeast**

**Yeast metabolomic study**

### Description

Two Saccharomyces Cerevisiae strains were compared under two different environmental conditions, 37 metabolites expression are measured.

### Usage

data(yeast)

### Format

A list containing the following components:

- **data**: data matrix with 55 rows and 37 columns. Each row represents an experimental sample, and each column a single metabolite.
- **strain**: a factor containing the type of strain (MT or WT).
- **condition**: a factor containing the type of environmental condition (AER or ANA).
- **strain.condition**: a crossed factor between strain and condition.

### Details

In this study, two Saccharomyces cerevisiae strains were used - wild-type (WT) and mutant (MT), and were carried out in batch cultures under two different environmental conditions, aerobic (AER) and anaerobic (ANA) in standard mineral media with glucose as the sole carbon source. After normalization and pre processing, the metabolomic data results in 37 metabolites and 55 samples which include 13 MT-AER, 14 MT-ANA, 15 WT-AER and 13 WT-ANA samples.
References

Index

*Topic algebra
  ipca, 47
  mat.rank, 54
  nipals, 92
  pca, 94
  sipca, 157
  spca, 159
  tune.pca, 190
*Topic cluster
  cim, 25
  cimDiablo, 33
  unmmap, 197
*Topic color
  color.jet, 38
*Topic datasets
  breast.TCGA, 23
  breast.tumors, 24
  diverse.165, 39
  Koren.165, 49
  linnerud, 50
  liver.toxicity, 51
  multidrug, 84
  nutrimouse, 93
  srbct, 170
  stemcells, 171
  vac18, 198
  vac18.simulated, 199
  yeast, 208
*Topic dplot
  image, 44
  imgCor, 45
  mixOmics, 80
  network, 87
  plotArrow, 111
  plotIndiv, 121
  plotVar, 135
  tune.mint.splsda, 183
  tune.rcc, 192
*Topic graphs
  cim, 25
  cimDiablo, 33
  network, 87
*Topic hplot
  cim, 25
  cimDiablo, 33
  image, 44
  mixOmics, 80
  network, 87
  plot.perf, 106
  plot.rcc, 109
  plot.tune.splsda, 110
  plotArrow, 111
  plotIndiv, 121
  plotVar, 135
*Topic  iplot
  cim, 25
  cimDiablo, 33
  network, 87
*Topic multivariate
  auroc, 4
  block.pls, 9
  block.plsda, 12
  block.spls, 15
  block.splsda, 19
  cim, 25
  cimDiablo, 33
  circoSPlot, 35
  explained_variance, 41
  imgCor, 45
  mint.block.pls, 55
  mint.block.plsda, 58
  mint.block.spls, 61
  mint.block.splsda, 64
  mint.pca, 67
  mint.pls, 69
  mint.plsda, 71
  mint.spls, 74
  mint.splsda, 77
<table>
<thead>
<tr>
<th>Topic</th>
<th>Terms</th>
</tr>
</thead>
<tbody>
<tr>
<td>mixOmics</td>
<td>80</td>
</tr>
<tr>
<td>network</td>
<td>87</td>
</tr>
<tr>
<td>nipals</td>
<td>92</td>
</tr>
<tr>
<td>perf</td>
<td>98</td>
</tr>
<tr>
<td>plot.perf</td>
<td>106</td>
</tr>
<tr>
<td>plot.rcc</td>
<td>109</td>
</tr>
<tr>
<td>plot.tune.splsda</td>
<td>110</td>
</tr>
<tr>
<td>plotArrow</td>
<td>111</td>
</tr>
<tr>
<td>plotContribution</td>
<td>115</td>
</tr>
<tr>
<td>plotDiablo</td>
<td>119</td>
</tr>
<tr>
<td>plotIndiv</td>
<td>121</td>
</tr>
<tr>
<td>plotLoadings</td>
<td>128</td>
</tr>
<tr>
<td>plotVar</td>
<td>135</td>
</tr>
<tr>
<td>pls</td>
<td>140</td>
</tr>
<tr>
<td>plsda</td>
<td>143</td>
</tr>
<tr>
<td>predict</td>
<td>145</td>
</tr>
<tr>
<td>print</td>
<td>150</td>
</tr>
<tr>
<td>rcc</td>
<td>152</td>
</tr>
<tr>
<td>spls</td>
<td>161</td>
</tr>
<tr>
<td>splsda</td>
<td>166</td>
</tr>
<tr>
<td>study_split</td>
<td>171</td>
</tr>
<tr>
<td>summary</td>
<td>172</td>
</tr>
<tr>
<td>tune</td>
<td>174</td>
</tr>
<tr>
<td>tune.block.splsda</td>
<td>178</td>
</tr>
<tr>
<td>tune.mint.splsda</td>
<td>183</td>
</tr>
<tr>
<td>tune.multilevel</td>
<td>185</td>
</tr>
<tr>
<td>tune.rcc</td>
<td>192</td>
</tr>
<tr>
<td>tune.splsda</td>
<td>193</td>
</tr>
<tr>
<td>vip</td>
<td>200</td>
</tr>
<tr>
<td>withinVariation</td>
<td>201</td>
</tr>
<tr>
<td>wrapper.rgcca</td>
<td>202</td>
</tr>
<tr>
<td>wrapper.sgcca</td>
<td>205</td>
</tr>
<tr>
<td>regression</td>
<td>4, 100, 102</td>
</tr>
<tr>
<td>utilities</td>
<td>86</td>
</tr>
<tr>
<td>*</td>
<td>113</td>
</tr>
<tr>
<td>arthritis</td>
<td>7, 124, 148</td>
</tr>
<tr>
<td>barplot</td>
<td>109, 130</td>
</tr>
<tr>
<td>biplot</td>
<td>97, 191</td>
</tr>
<tr>
<td>block.pls</td>
<td>9, 14, 18, 83, 131, 148</td>
</tr>
<tr>
<td>block.plsda</td>
<td>11, 12, 22, 83, 131, 145, 148</td>
</tr>
<tr>
<td>block.spls</td>
<td>11, 15, 22, 83, 131, 148, 206</td>
</tr>
<tr>
<td>block.splsda</td>
<td>14, 18, 19, 37, 83, 120, 131, 148, 168, 181</td>
</tr>
<tr>
<td>breast.TCGA</td>
<td>23</td>
</tr>
<tr>
<td>breast.tumors</td>
<td>24</td>
</tr>
<tr>
<td>cim</td>
<td>25, 35, 41, 90, 117, 137, 154, 164, 168, 202</td>
</tr>
<tr>
<td>cimDiablo</td>
<td>33</td>
</tr>
<tr>
<td>circosPlot</td>
<td>35</td>
</tr>
<tr>
<td>color</td>
<td>38</td>
</tr>
<tr>
<td>color.GreenRed</td>
<td>90</td>
</tr>
<tr>
<td>color.Red</td>
<td>38</td>
</tr>
<tr>
<td>color.RDC</td>
<td>38</td>
</tr>
<tr>
<td>color.RDC150</td>
<td>38</td>
</tr>
<tr>
<td>color.rG</td>
<td>38</td>
</tr>
<tr>
<td>color.RDC</td>
<td>38</td>
</tr>
<tr>
<td>colors</td>
<td>38</td>
</tr>
<tr>
<td>cor</td>
<td>46</td>
</tr>
</tbody>
</table>
dist, 27
diverse, 165, 39

eigen, 93
estim.regul, 41
explained.variance, 41

get.BER (get.confusion_matrix), 42
get.confusion_matrix, 42
gray, 38

hclust, 27, 29, 35
heat.colors, 38, 44
heatmap, 29, 35
hsv, 38

image, 28, 44, 44, 46
image.estim.regul, 45
image.tune.rcc, 45, 193
imgCor, 45
ipca, 47, 158
Koren, 165, 49

layout, 28
linnerud, 50
liver.toxicity, 51
logratio.transfo, 52, 96, 141, 144, 160, 163, 167, 191

map, 53
mat.rank, 54
mint.block.pls, 55, 63, 67, 83, 131, 148
mint.block.plsda, 57, 58, 60, 63, 67, 83, 131, 145, 148
mint.block.spls, 57, 60, 61, 67, 83, 131, 148
mint.block.splslsda, 57, 60, 63, 64, 83, 131, 148, 168
mint.pca, 67
mint.pls, 69, 73, 76, 79, 83, 131, 148, 172
mint.spls, 69, 71, 73, 74, 83, 131, 148, 172
mint.splslsda, 69, 71, 73, 76, 77, 83, 131, 148, 172, 184
mixOmics, 80
multidrug, 84

nearZeroVar, 10, 13, 16, 21, 56, 59, 62, 65, 70, 72, 75, 78, 82, 86, 141, 143, 162, 167, 176, 180, 183, 187, 194, 204, 206
network, 29, 35, 41, 87, 137, 154, 164, 168, 202
nipals, 54, 92, 97, 102, 191
nutrimouse, 93
order.dendrogram, 28
palette, 38
par, 27, 34, 46, 109, 136, 137
pca, 48, 53, 94, 158, 160
catune, 97
perf, 5, 11, 14, 18, 22, 57, 60, 63, 67, 69, 71, 73, 76, 79, 98, 107, 108, 142, 145, 164, 168
plot.perf, 102, 106
plot.rcc, 109, 154
plot.tune (plot.tune.splsda), 110
plot.tune.splslsda, 110
plotArrow, 11, 14, 18, 22, 111
plotContrib, 115
plotDiablo, 119
plotIndiv, 7, 8, 11, 14, 18, 22, 41, 48, 57, 60, 63, 67–73, 75, 76, 78, 79, 97, 121, 142, 145, 154, 158, 164, 168, 191, 202, 204, 207
plotLoadings, 11, 14, 18, 22, 68, 70, 72, 75, 78, 117, 128
plotVar, 11, 14, 18, 22, 29, 35, 41, 48, 57, 60, 63, 67–73, 75, 76, 78, 79, 90, 97, 117, 135, 142, 145, 154, 158, 164, 168, 191, 202, 204, 207
points, 109, 112, 113, 123, 124, 136
polygon, 8
prcomp, 93, 97
predict, 7, 8, 11, 14, 18, 22, 43, 57, 60, 63, 67, 69, 71, 73, 76, 79, 99, 102, 142, 145, 145, 164, 168, 183
predict.splslsda, 196
princomp, 93, 95, 191
print, 150
rainbow, 26, 34, 38, 44
rcc, 151, 152, 153, 174
scale, 27, 95, 159, 176, 190
INDEX

select.var (selectVar), 154
selectVar, 11, 14, 18, 22, 154
sipca, 48, 157
spca, 159
spls, 41, 53, 57, 60, 63, 67, 69, 71, 73, 76, 79,
      83, 87, 108, 131, 142, 148, 151, 161,
      168, 174, 200, 202
splsda, 41, 53, 83, 87, 108, 124, 131, 145,
      148, 166, 196, 202
srbct, 170
stemcells, 171
study_split, 171
summary, 57, 60, 63, 67, 69, 71, 73, 76, 79,
      142, 145, 154, 164, 168, 172, 200
svd, 93

terrain.colors, 26, 34, 38, 44
text, 113, 124
topo.colors, 26, 34, 38, 44
tune, 5, 174
tune.block.splsda, 21, 178
tune.mint.splsda, 78, 110, 178, 183
tune.multilevel, 178, 185
tune.pca, 97, 178, 190
tune.rcc, 41, 44, 153, 154, 178, 192
tune.splsda, 110, 178, 188, 193
tune.splslevel, 178
tune.splslevel (tune.multilevel), 185

unmap, 53, 197, 204, 206

vac18, 198
vac18.simulated, 199
vip, 151, 174, 200

withinVariation, 96, 141, 144, 160, 163,
      167, 191, 201
wrapper.rgccca, 202, 204, 207
wrapper.sgcca, 204, 205, 207
wrapper.sgccda (block.splsda), 19

xyplot, 107, 108, 110

yeast, 208