Package ‘bootsPLS’

May 17, 2018

Type Package

Title Bootstrap Subsamplings of Sparse Partial Least Squares - Discriminant Analysis for Classification and Signature Identification

Version 1.1.2

Author Florian Rohart [aut, cre], Kim-Anh Le Cao [boss], Christine Wells [boss]

Maintainer Florian Rohart <florian.rohart@gmail.com>

Description
Applicable to any classification problem with more than 2 classes. It relies on bootstrap subsamplings of sPLS-DA and provides tools to select the most stable variables (defined as the ones consistently selected over the bootstrap subsamplings) and to predict the class of test samples.

License GPL-3

Depends mixOmics (>= 6.3.1)

URL http://florian.rohart.free.fr/Professional_page/Publications.html

BugReports https://bitbucket.org/FlorianR/package-bootspls/issues

NeedsCompilation no

Repository CRAN

Date/Publication 2018-05-17 03:40:17 UTC

R topics documented:

`bootsPLS-package` *Bootstrap Subsamplings of Sparse Partial Least Squares - Discriminant Analysis for Classification and Signature Identification*

Description

Applicable to any classification problem with more than 2 classes. It relies on bootstrap subsamplings of sPLS-DA and provides tools to select the most stable variables (defined as the ones consistently selected over the bootstrap subsamplings) and to predict the class of test samples.

Details

The DESCRIPTION file:

Package:	bootsPLS
Type:	Package
Title:	Bootstrap Subsamplings of Sparse Partial Least Squares - Discriminant Analysis for Classification and Signatu
Version:	1.1.2
Author:	Florian Rohart [aut, cre], Kim-Anh Le Cao [boss], Christine Wells [boss]
Maintainer:	Florian Rohart <florian.rohart@gmail.com>
Description:	Applicable to any classification problem with more than 2 classes. It relies on bootstrap subsamplings of sPLS-
License:	GPL-3
Depends:	mixOmics (>= 6.3.1)
Packaged:	14-12-2017; Florian
URL:	http://florian.rohart.free.fr/Professional_page/Publications.html
BugReports:	https://bitbucket.org/FlorianR/package-bootspls/issues

Index of help topics:

```
CI.prediction           Compute Confidence Intervals (CI) for test
                        samples
MSC                     Mesenchymal Stem Cells data
bootsPLS                Performs replications of sPLSDA on random
                        subsamplings of the data
bootsPLS-package        Bootstrap Subsamplings of Sparse Partial Least
                        Squares - Discriminant Analysis for
                        Classification and Signature Identification
compile.bootsPLS.object
                        Combine several bootsPLS objects into one
component.selection     Tune the number of components
```

```
fit.model                Create a spls.constraint object by fitting a
                         constraint spls on a bootsPLS object
plot.bootsPLS            Plot the frequency of selection of all
                         variables for all the PLS-component.
plot.component.selection
                         Plot the results of the testing procedure to
                         determine the number of component to select
plot.predictCI           Plot confidence Intervals
plot.variable.selection
                         Plot the results of the testing procedure to
                         determine the number of variables to select
prediction               prediction
spls.hybrid              spls.hybrid, midway between PLS and sPLS
variable.selection       Tune the number of variables on each component
```

The package implements the methodology described in Rohart *et al.* (2016) for identifying genes that differentiate Mesemchymal Stromal Cells from other cell types. The method is applicable to any classification problem with more than 2 classes. It relies on bootstrap subsamplings of sPLS-DA and provides tools to select the most stable variables (defined as the ones consistently selected over the bootstrap subsamplings) and to predict the class of test samples.

Three major functions:
*`bootsPLS` performs a `tune.splsda` of the `mixOmics` package on several random subsamplings. It records the selected variables on each replication. The `compile.bootsPLS.object` function is to be used if several calls to `bootsPLS` are made.
*`fit.model` fits a constraint spls (see `spls.hybrid`) on the most stable variables. The variables can be automatically selected by setting auto.tune=TRUE.
*`prediction` predicts the class of test samples and compute Confidence Interval (CI) of the prediction values.

Author(s)

Florian Rohart [aut, cre], Kim-Anh Le Cao [boss], Christine Wells [boss]

Maintainer: Florian Rohart <florian.rohart@gmail.com>

References

Rohart *et al.* (2016). A Molecular Classification of Human Mesenchymal Stromal Cells. PeerJ, DOI 10.7717/peerj.1845
Le Cao, K.-A., Boitard, S. and Besse, P. (2011). Sparse PLS Discriminant Analysis: biologically relevant feature selection and graphical displays for multiclass problems. *BMC Bioinformatics* **12**:253.

Examples

```
## Not run:
data(MSC)
```

```
X=MSC$X
Y=MSC$Y


boot1=bootsPLS(X=X,Y=Y,ncomp=3,many=5,kCV=5)
boot2=bootsPLS(X=X,Y=Y,ncomp=3,many=5,kCV=5,showProgress=FALSE)
boot3=bootsPLS(X=X,Y=Y,ncomp=3,many=5,kCV=5,showProgress=FALSE, cpus=2)

# construct a list of bootsPLS object
bootsPLS.object=list(boot1,boot2,boot3)

# compile the outputs in one bootsPLS object
boot=compile.bootsPLS.object(bootsPLS.object)

# fit the model
fit=fit.model(boot,auto.tune=TRUE) #tuning number of components and variables
fit=fit.model(boot,ncomp=2) #tuning number of variables on the 2components
plotIndiv(fit,ind.names=FALSE, legend=TRUE)

# prediction and Confidence Interval, here we use the same dataset but it should be external data
pred=prediction(fit,X.test=X,CI=TRUE)

head(pred$Y.hat.test[,,"comp.1"])
lapply(pred$out.CI$CI$'comp.1',head)


## End(Not run)
```

`bootsPLS` *Performs replications of sPLSDA on random subsamplings of the data*

Description

Performs replications of sPLSDA on random subsamplings of the data

Usage

```
bootsPLS(X,Y,near.zero.var,many=50,ncomp=2,
            dist = c("max.dist", "centroids.dist", "mahalanobis.dist"),
            save.file,ratio,kCV=10,grid,cpus,nrepeat=1,showProgress=TRUE)
```

Arguments

X	Input matrix of dimension n * p; each row is an observation vector.
Y	Factor with at least q>2 levels.
near.zero.var	Logical. If TRUE, a pre-screening step is performed to remove predictors with near-zero variance. See nearZeroVar.
many	How many replications of the sPLS-DA analysis are to be done?

`ncomp`	How many component are to be included in the sPLS-DA analysis?
`dist`	Indicates the distance that is used to classify the samples. One of "max.dist", "centroids.dist", "mahalanobis.dist". Default is "max.dist"
`save.file`	If the outputs are to be saved, this argument allows you to do it at the end of each replication. A full path is expected. Convenient if you run this function on a cluster and it is killed before completion, e.g. due to a too short requested time.
`ratio`	Number between 0 and 1. It is the proportion of the n samples that are put aside and considered as an internal testing set. The (1-ratio)*n samples are used as a training set and the `kCV` fold cross validation is performed on them. Default is 0.3
`kCV`	Number of fold for the cross validation. Default is 10.
`grid`	A vector of value for the tuning of the `keepX` parameter of sPLS-DA on each component. See `spls` for more details on `keepX`. Default is `grid=1:min(40,ncol(X))`.
`cpus`	Number of cpus to use when running the code in parallel.
`nrepeat`	Number of times the Cross-Validation process is repeated for each of the `many` replications. See `tune.splsda` for details.
`showProgress`	Logical. If TRUE, shows the progress of the algorithm. It also gives a list of which variables are selected on each component.

Details

Performs replication of `tune.splsda` on random subsamplings of the data and record which variables are selected on which subsamplings. It also gives a confusion matrix for each component and for each subsamplings.

Value

A 'bootsPLS' object is returned for which `plot`, `fit.model` and `prediction` are available.

`ClassifResult`	A 4-dimensional array. The two first dimensions consists in the confusion matrix. The third dimension is relative to the number of components `ncomp`. The fourth dimension concerns the number of replication `many`.
`loadings.X`	A 3-dimensional array. Loadings vector of X, for each component and each replication.
`selection.variable`	A 3-dimensional array. Gives the selected variables for each component and each replication. It is obtained by replacing each non zero value in `loadings.X` by 1.
`frequency`	A matrix of size ncomp*p. Gives the frequency of selection for each variable on each component. It is obtained as a mean over the third dimension of `selection.variable`
`nbr.var`	Matrix of size many*ncomp. Gives the number of variables that have been selected on each component for each replication.

`learning.sample`	Matrix of size n*many. Gives the samples that have been used in the internal training set over the `many` replications. These samples have the value 1, the others 0.
`prediction`	A 3-dimensional array of size n*many*ncomp. Gives the prediction for the chosen `dist` of all the samples, either in the learning set or the testing set.
`data`	A list of the input data X, Y and of the distance used to classify the sample ("max.dist", "centroids.dist" or "mahalanobis.dist").

References

Rohart *et al.* (2016). A Molecular Classification of Human Mesenchymal Stromal Cells. PeerJ, DOI 10.7717/peerj.1845

See Also

`splsda`, `plot.bootsPLS`, `fit.model`, `prediction`

Examples

```
## Not run:
data(MSC)
X=MSC$X
Y=MSC$Y
dim(X)
table(Y)


boot=bootsPLS(X=X,Y=Y,ncomp=3,many=5,kCV=5)


# saving the outputs in a Rdata file, the file is saved after each iteration
# if used on a cluster, you can use the `cpus' argument as well
save.file=paste(getwd(),"/MSC.",Sys.getpid(),".Rdata",sep="")
boot=bootsPLS(X=X,Y=Y,ncomp=3,many=5,kCV=5,save.file=save.file)


## End(Not run)
```

`CI.prediction` *Compute Confidence Intervals (CI) for test samples*

Description

Compute Confidence Intervals (CI) for test samples based on random subsamplings

Usage

```
CI.prediction(object,X,Y,signature,ncomp,many,
            subsampling.matrix,ratio,X.test,level.CI,save.file)
```

Arguments

`object`	a ‘spls.constraint’ object, as one resulting from `fit.model`. If object is missing: X, Y, signature are needed.
`X`	Only used if `object` is missing. Input matrix of dimension n * p; each row is an observation vector.
`Y`	Only used if `object` is missing. Factor with at least q>2 levels.
`signature`	Only used if `object` is missing. A list containing which variables are to be kept on each component.
`ncomp`	Only used if `object` is missing. How many component do you want to include in the sPLS-DA analysis?
`many`	How many subsamplings do you want to do? Default is 100
`subsampling.matrix`	Optional matrix of `many` columns. Gives the samples to subsample as an internal learning set.
`ratio`	Number between 0 and 1. It is the proportion of the n samples that are put aside and considered as an internal testing set. The (1-ratio)*n samples are used as a training set and the `kCV` fold cross validation is performed on them. Default is 0.3
`X.test`	Test matrix.
`level.CI`	A 1- `level.CI`% confidence interval is calculated.
`save.file`	Save the outputs of the functions in `save.file.Rdata`.

Details

This function can work with a ‘spls.constraint’ object or with the input data (X, Y, signature). See examples below to see the difference in use.

Value

`CI`	A (1- `level.CI`)% confidence interval is returned for each samples in `X.test`
`Y.hat.test`	A four dimensional array. The two first dimensions are an estimation of the dummy matrix obtained from Y (size n * number of sample types). The third dimension is relative to the number of components `ncomp`. The fourth dimension concerns the number of subsamplings.
`ClassifResult`	A 5-dimensional array. The two first dimensions consists in the confusion matrix. The third dimension is relative to the number of components `ncomp`. The fourth dimension concerns the number of subsamplings. The fifth and last dimension is relative to the different distances "max.dist", "centroids.dist" and "mahalanobis.dist".

`loadings.X`	A 3-dimensional array. Loadings vector of X, for each component and each subsampling.
`prediction.X`	A 4-dimensional array of size n*many*ncomp*3. Gives the prediction for the chosen `method` of all the samples, either in the internal learning set or the internal testing set. The last dimension is relative to the different distances "max.dist", "centroids.dist" and "mahalanobis.dist".
`prediction.X.test`	A 4-dimensional array of size nrow(X.test)*many*ncomp*3. Gives the prediction for the chosen `method` of all the test samples in X.test. The last dimension is relative to the different distances "max.dist", "centroids.dist" and "mahalanobis.dist".
`learning.sample`	Matrix of size n*many. Gives the samples that have been used in the internal training set over the `many` replications. These samples have the value 1, the others 0.
`coeff`	A list of means.X, sigma.X, means.Y and sigma.Y. Means and variances for the variables of X and the columns of the dummy matrix obtained from Y, each row is a subsampling.
`data`	A list of the input data X, Y and of signature, which is a list containing the variables kept on each component.

See Also

`fit.model`, `prediction`

Examples

```
## Not run:
data(MSC)
X=MSC$X
Y=MSC$Y


# with a bootsPLS object
boot=bootsPLS(X=X,Y=Y,ncomp=3,many=5,kCV=5)
fit=fit.model(boot,ncomp=3)

CI=CI.prediction(fit)
CI=CI.prediction(fit,X.test=X)
plot(CI)

lapply(CI$CI$'comp.1',head)
lapply(CI$CI$'comp.2',head)
lapply(CI$CI$'comp.3',head)


# without a spls.constraint object. X,Y and signature are needed
# the results should be similar
#(not the same because of the random subsamplings,
# exactly the same if subsampling.matrix is an input)
```

```
signature=fit$data$signature
CI=CI.prediction(X=X,Y=Y,signature=signature)
CI=CI.prediction(X=X,Y=Y,signature=signature,X.test=X)

lapply(CI$CI$'comp.1',head)
lapply(CI$CI$'comp.2',head)
lapply(CI$CI$'comp.3',head)

## End(Not run)
```

---

compile.bootsPLS.object

Combine several bootsPLS objects into one

Description

Combine several bootsPLS objects into one

Usage

```
compile.bootsPLS.object(bootsPLS.list,path,pattern,file,save.file)
```

Arguments

`bootsPLS.list`	A list of bootsPLS object, as obtained from multiple calls to `bootsPLS`
`path`	Only used if `bootsPLS.list` is missing. A path to the Rdata files containing the bootsPLS outputs to combine. Will be passed in the `list.files` function.
`pattern`	Only used if `bootsPLS.list` is missing. A commun pattern to the Rdata files to combine. Will be passed in the `list.files` function.
`file`	Only used if `bootsPLS.list` is missing. Vector of Rdata files to combine. Will be loaded as `path/file[i]`
`save.file`	Optional. Full path of the Rdata file to be saved with the combined objects.

Details

This function works with either a list of `bootsPLS` object, a `path` and a `pattern` or a `path` and a `file`. The outputs can be saved into a Rdata file. See Examples below.

Value

A 'bootsPLS' object is returned for which `plot`, `fit.model` and `prediction` are available. See `bootsPLS` for the outputs.

See Also

`plot.bootsPLS`, `fit.model`, `prediction`, `bootsPLS`

Examples

```
## Not run:
data(MSC)
X=MSC$X
Y=MSC$Y
dim(X)
table(Y)

# perform several bootsPLS analysis
boot1=bootsPLS(X=X,Y=Y,ncomp=3,many=5,kCV=5)
boot2=bootsPLS(X=X,Y=Y,ncomp=3,many=5,kCV=5,showProgress=FALSE)
boot3=bootsPLS(X=X,Y=Y,ncomp=3,many=5,kCV=5,showProgress=FALSE)

# construct a list of bootsPLS object
bootsPLS.object=list(boot1,boot2,boot3)

# compile the outputs in one bootsPLS object
boot=compile.bootsPLS.object(bootsPLS.object)


# ======================================
# all can work from the saved file:
# ======================================
#convenient if used on a cluster

# perform several bootsPLS analysis and save the outputs
boot1=bootsPLS(X=X,Y=Y,ncomp=3,many=5,kCV=5,save.file="MSC1.Rdata")
boot2=bootsPLS(X=X,Y=Y,ncomp=3,many=5,kCV=5,showProgress=FALSE,save.file="MSC2.Rdata")
boot3=bootsPLS(X=X,Y=Y,ncomp=3,many=5,kCV=5,showProgress=FALSE,save.file="MSC3.Rdata")

# compile the outputs in one bootsPLS object
boot=compile.bootsPLS.object(path=paste(getwd(),"/",sep=""),
    pattern="MSC",save.file="MSC.all.Rdata")

# or
boot=compile.bootsPLS.object(path=paste(getwd(),"/",sep=""),
    file=c("MSC1.Rdata","MSC2.Rdata","MSC3.Rdata"),save.file="MSC.all.Rdata")



## End(Not run)
```

component.selection *Tune the number of components*

Description

Performs a multiple hypotheses testing procedure to choose the number of components in the splsda analysis

Usage

```
component.selection(object,alpha,showProgress=TRUE)
```

Arguments

`object`	a ‘bootsPLS’ object, as obtained from `bootsPLS`
`alpha`	Level of the test
`showProgress`	Logical. If TRUE, shows the progress of the algorithm.

Details

The testing procedure evaluates the gain in classification accuracy when a new PLS-component is added. This is done by on-sided t-test of level `alpha` applied on the classification accuracy obtained in the ‘bootsPLS’ object. See the reference below for more details on the multiple testing procedure.

Value

A ’component.selection’ object is returned for which `plot` is available.

`pval`	pvalue obtained from the testing procedure.
`opt`	Number of components chosen by the procedure
`object`	input ‘bootsPLS’ object
`alpha`	input level of the test

References

Rohart *et al.* (2016). A Molecular Classification of Human Mesenchymal Stromal Cells. PeerJ, DOI 10.7717/peerj.1845

See Also

`plot.component.selection`, `fit.model`, `prediction`

Examples

```
## Not run:
data(MSC)
X=MSC$X
Y=MSC$Y
dim(X)
table(Y)

boot=bootsPLS(X=X,Y=Y,ncomp=3,many=5,kCV=5)

comp=component.selection(boot)

## End(Not run)
```

`fit.model` *Create a spls.constraint object by fitting a constraint spls on a bootsPLS object*

Description

Create a spls.constraint object by fitting a constraint spls with the `spls.hybrid` function on a 'bootsPLS' object

Usage

```
fit.model(object,auto.tune,X,Y,ncomp,signature,alpha,limit,showProgress=TRUE)
```

Arguments

`object`	a 'bootsPLS' object', as obtained from `bootsPLS`.
`auto.tune`	Logical. If TRUE, tune the optimal number of component (ncomp) and which variables to select on each component (signature). It only works with `object`
`X`	Input matrix of dimension n * p; each row is an observation vector.
`Y`	Factor with at least q>2 levels.
`ncomp`	How many component are to be included in the sPLS-DA analysis?
`signature`	A list containing which variables to keep on each component.
`alpha`	Level of the test.
`limit`	Vector of maximal number of genes to include on each component.
`showProgress`	Logical. If TRUE, shows the progress of the algorithm.

Details

This function fit a `spls.hybrid` on the variables included in `signature`, which can be an input or internally calculated by setting `auto.tune=TRUE`. If `object` is given as an input, (X, Y) are ignored. If `auto.tune=TRUE`, `ncomp, signature` are ignored.

Value

A 'spls.constraint' object is returned for which `plotIndiv` is available.

The outputs are the ones from `spls.hybrid`, plus

`data`	A list of the input data X, Y, Y.mat (dummy matrix) and of `signature`, which is a list containing the variables kept on each component. If a tuning occurs, either for the number of components or for the variables or both, outputs $component.selection and $variable.selection are available; see `component.selection` and `variable.selection`.

See Also

prediction, CI.prediction, plotIndiv

Examples

```
## Not run:
data(MSC)
X=MSC$X
Y=MSC$Y


boot=bootsPLS(X=X,Y=Y,ncomp=3,many=5,kCV=5)

# with a bootsPLS object and auto.tune=TRUE
fit=fit.model(boot,auto.tune=TRUE)
plot(fit$component.selection)
plot(fit$variable.selection)

# with a bootsPLS object and ncomp=2
fit=fit.model(boot,ncomp=2)

# with a bootsPLS object and ncomp/signature as input
signature=fit$data$signature
fit=fit.model(boot,ncomp=2,signature=signature)

# with no bootsPLS object
fit=fit.model(X=X,Y=Y,ncomp=2,signature=signature)# bootsPLS object

plotIndiv(fit,ind.names=FALSE, legend=TRUE)


## End(Not run)
```

MSC *Mesenchymal Stem Cells data*

Description

The data is a subset of the data contained in Rohart et al. (2015). It contains 150 samples, either Mesenchymal Stem Cells (MSC) or Non-MSC, and 200 genes.

Usage

```
data(MSC)
```

Format

A list containing the following components:

- `X` data matrix with 150 rows and 200 columns. Each row represents an experimental sample, and each column a single gene.
- `Y` a vector containing the sample type (MSC or Non-MSC).

Details

This data is a subset of a bigger study; the samples and genes have been randomly selected for this package. All samples have been background corrected, log2 transformed and YuGene normalised.

References

Rohart *et al.* (2016). A Molecular Classification of Human Mesenchymal Stromal Cells. PeerJ, DOI 10.7717/peerj.1845
Kim-Anh Le Cao, Florian Rohart, Leo McHugh, Othmar Korn, Christine A. Wells. YuGene: A simple approach to scale gene expression data derived from different platforms for integrated analyses. Genomics. http://dx.doi.org/10.1016/j.ygeno.2014.03.001.

`plot.bootsPLS` *Plot the frequency of selection of all variables for all the PLS-component.*

Description

Plot the frequency of selection of all variables for all the PLS-component.

Usage

```
## S3 method for class 'bootsPLS'
plot(x,light,pch,col,legend.position,title, name.var=TRUE,...)
```

Arguments

`x`	Object of class ‘bootsPLS’.
`light`	Between 0 and 1. A lot of variables usually have a low frequency, this argument allows you to discard those from the plot. Only the variables that have a frequency higher than `light` on at least one component are plotted.
`pch`	A vector of length ncomp (argument of `bootsPLS`). See `par` for details.
`col`	A vector of length ncomp (argument of `bootsPLS`). See `par` for details.
`legend.position`	Location of the legend, see `legend` for more details. If set to FALSE, no legend is plotted.
`title`	title of the plot

`name.var`	If TRUE, add the name of the variables in `x$data$X` on the x-axis. A vector of length the number of variables can be supplied for customized names.
`...`	not used

Details

Plot the frequency of selection of all variables depending on the component. By default, a legend and a title are added to the plot, those can be changed/removed by changing the respective inputs.

See Also

`bootsPLS`

Examples

```
## Not run:
data(MSC)
X=MSC$X
Y=MSC$Y
dim(X)
table(Y)


boot=bootsPLS(X=X,Y=Y,ncomp=3,many=5,kCV=5)

plot(boot)
plot(boot,light=0.3,legend.position=FALSE)
plot(boot,light=0.3,legend.position=FALSE, name.var = paste("Gene",1:ncol(X)))

## End(Not run)
```

plot.component.selection

Plot the results of the testing procedure to determine the number of component to select

Description

Plot the results of the testing procedure to determine the number of component to select

Usage

```
## S3 method for class 'component.selection'
plot(x,pch,col,type,lty,...)
```

Arguments

`x`	Object of class "component.selection".
`pch`	See `par` for details.
`col`	See `par` for details.
`type`	See `par` for details.
`lty`	See `par` for details.
`...`	not used

Details

This function plots the p-value of the one sided t-test with respect to the number of components. See `component.selection` and the reference below for more details on the multiple testing procedure

References

Rohart *et al.* (2016). A Molecular Classification of Human Mesenchymal Stromal Cells. PeerJ, DOI 10.7717/peerj.1845

See Also

`bootsPLS`, `component.selection`

Examples

```
## Not run:
data(MSC)
X=MSC$X
Y=MSC$Y


boot=bootsPLS(X=X,Y=Y,ncomp=3,many=5,kCV=5)
comp=component.selection(boot)

plot(comp)


## End(Not run)
```

`plot.predictCI` *Plot confidence Intervals*

Description

Display the confidence intervals for the prediction values of a specific level of the outcome

Usage

```
## S3 method for class 'predictCI'
plot(x,
ncomp=1, level=1, las=2, col, title, name.var = TRUE, abline=TRUE,...)
```

Arguments

`x`	a 'predict.CI' object, as one resulting from `CI.prediction`, or from `prediction$out.CI`.
`ncomp`	Component to be plotted
`level`	One of the levels of the outcome to plot
`las`	las argument for the x-axis labels
`col`	Color for the confidence Intervals. Can be a single value or a different value for each sample.
`title`	title of the plot
`name.var`	If TRUE, add the name of the variables on the x-axis. A vector of length the number of variables can be supplied for customized names.
`abline`	logical. Add two horizontal lines: 0.5 and 1/the number of levels of the outcome. Note that if the outcome only has two levels, only one line is plotted.
`...`	not used

Details

Plot the Confidence Intervals for a specific component and a specific level. In the special case of the "max.dist" being used (in the `bootsPLS` function), a line at 0.5 and 1/number of levels is added to the plot with `abline=TRUE`. These lines highlight the predicted class of the samples. For example if the level 1 out of 2 levels is displayed: any sample above 0.5 is predicted as level 1, any samples below 0.5 is predicted as level 2, and samples with a confidence interval overlaying 0.5 are not a clear cut. Another example if the level 1 out of 3 levels is displayed: any sample above 0.5 is predicted as level 1, any samples below 1/3 is predicted as not level 1, and samples in between 1/3 and 0.5 can't be decided from this graph only (you can display other levels, or find the predicted class in your object). See example below.

In the case of another distance being used, the lines should not be added to the graph `abline=FALSE`.

See Also

`CI.prediction`, `prediction`

Examples

```
## Not run:
data(MSC)
X=MSC$X
Y=MSC$Y


# with a bootsPLS object
boot=bootsPLS(X=X,Y=Y,ncomp=3,many=5,kCV=5)
```

```
fit=fit.model(boot,ncomp=3)

CI=CI.prediction(fit,X.test=X)
plot(CI)

pred=prediction(fit,X.test=X, CI=TRUE)
plot(pred$out.CI, ncomp=2)
# we color each confidence interval by the predicted class
plot(pred$out.CI, ncomp=2,
    col = color.mixo(factor(pred$predicted.test$"max.dist"[,2]))) #second component


# because bootsPLS was used with dist="max.dist" (by default)
# and because there are only two levels in the outcome,
# everything above the 0.5 line is predicted as MSC

par(mfrow=c(3,1))
plot(pred$out.CI, ncomp=1, level="MSC")
plot(pred$out.CI, ncomp=2, level="MSC")
plot(pred$out.CI, ncomp=3, level="MSC")


#we can do the same things for the second level (Non-MSC)
par(mfrow=c(3,1))
plot(pred$out.CI, ncomp=1, level="Non-MSC")
plot(pred$out.CI, ncomp=2, level="Non-MSC")
plot(pred$out.CI, ncomp=3, level="Non-MSC")


## End(Not run)
```

---

plot.variable.selection

Plot the results of the testing procedure to determine the number of variables to select

---

Description

Plot the results of the testing procedure to determine the number of variables to select on each component

Usage

```
## S3 method for class 'variable.selection'
plot(x,pch,col,type,lty,...)
```

Arguments

x	Object of class "variable.selection".

`pch`	A vector of length ncomp (argument of `bootsPLS`). See `par` for details.
`col`	A vector of length ncomp (argument of `bootsPLS`). See `par` for details.
`type`	A vector of length ncomp (argument of `bootsPLS`). See `par` for details.
`lty`	A vector of length ncomp (argument of `bootsPLS`). See `par` for details.
`...`	not used

Details

This function plots the p-value of the one sided t-test with respect to the frequency of each variable. See `variable.selection` and the reference below for more details on the multiple testing procedure.

References

Rohart *et al.* (2016). A Molecular Classification of Human Mesenchymal Stromal Cells. PeerJ, DOI 10.7717/peerj.1845

See Also

`bootsPLS`, `variable.selection`

Examples

```
## Not run:
data(MSC)
X=MSC$X
Y=MSC$Y


boot=bootsPLS(X=X,Y=Y,ncomp=3,many=5,kCV=5)
out=variable.selection(boot,ncomp=2,limit=c(40,40))
plot(out)



## End(Not run)
```

`prediction` *prediction*

Description

prediction

Usage

```
prediction (object,X,Y,signature,ncomp,X.test,CI,many,
                subsampling.matrix,ratio,level.CI,save.file)
```

Arguments

`object`	a 'spls.constraint' object, as one resulting from `fit.model`. If object is missing: X, Y, signature are needed.
`X`	Only used if `object` is missing. Input matrix of dimension n * p; each row is an observation vector.
`Y`	Only used if `object` is missing. Factor with at least q>2 levels; Response variable of length n, indicating the sample types.
`signature`	Only used if `object` is missing. A list containing which variables are to be kept on each component.
`ncomp`	Only used if `object` is missing. How many component do you want to include in the sPLS-DA analysis?
`X.test`	Test matrix.
`CI`	logical. If TRUE, the confidence interval are calculated.
`many`	How many subsamplings do you want to do? Default is 100
`subsampling.matrix`	Optional matrix of `many` columns. Gives the samples to subsample as an internal learning set.
`ratio`	Number between 0 and 1. It is the proportion of the n samples that are put aside and considered as an internal testing set. The (1-ratio)*n samples are used as a training set.
`level.CI`	A 1- `level.CI`% confidence interval is calculated.
`save.file`	Save the outputs of the functions in `save.file.Rdata`.

Details

This function can work with a `spls.constraint` object or with the input data (X, Y, signature). See examples below to see the difference in use.

Value

`CI`	A (1- `level.CI`)% confidence interval is returned for each samples in `X.test`
`Y.hat.test`	A four dimensional array. The two first dimensions are an estimation of the dummy matrix obtained from Y (size n * number of sample types). The third dimension is relative to the number of components `ncomp`. The fourth dimension concerns the number of subsamplings.
`ClassifResult`	A 5-dimensional array. The two first dimensions consists in the confusion matrix. The third dimension is relative to the number of components `ncomp`. The fourth dimension concerns the number of subsamplings. The fifth and last dimension is relative to the different distances "max.dist", "centroids.dist" and "mahalanobis.dist".
`loadings.X`	A 3-dimensional array. Loadings vector of X, for each component and each subsampling.

`prediction.X`	A 4-dimensional array of size n*many*ncomp*3. Gives the prediction for the chosen `method` of all the samples, either in the internal learning set or the internal testing set. The last dimension is relative to the different distances "max.dist", "centroids.dist" and "mahalanobis.dist".
`prediction.X.test`	A 4-dimensional array of size nrow(X.test)*many*ncomp*3. Gives the prediction for the chosen `method` of all the test samples in X.test. The last dimension is relative to the different distances "max.dist", "centroids.dist" and "mahalanobis.dist".
`learning.sample`	Matrix of size n*many. Gives the samples that have been used in the internal training set over the `many` replications. These samples have the value 1, the others 0.
`coeff`	A list of means.X, sigma.X, means.Y and sigma.Y. Means and variances for the variables of X and the columns of the dummy matrix obtained from Y, each row is a subsampling.
`data`	A list of the input data X, Y and of ind.kept.X, which is a list containing the variables kept on each component.

See Also

`fit.model`, `CI.prediction`

Examples

```
## Not run:
data(MSC)
X=MSC$X
Y=MSC$Y


# with a bootsPLS object
boot=bootsPLS(X=X,Y=Y,ncomp=3,many=5,kCV=5)
fit=fit.model(boot,ncomp=3)

# with a spls.constraint object and without CI
pred=prediction(fit,X.test=X)

lapply(pred$predicted.test,head)

# with a spls.constraint object and with CI
pred.CI=prediction(fit,X.test=X,CI=TRUE)
lapply(pred.CI$out.CI$CI$'comp.1',head)
lapply(pred.CI$out.CI$CI$'comp.2',head)
lapply(pred.CI$out.CI$CI$'comp.3',head)

# without a spls.constraint object. X,Y and signature are needed
# the results should be similar
#(not the same because of the random subsamplings,
# exactly the same if subsampling.matrix is an input)
```

```
signature=fit$data$signature
pred=prediction(X=X,Y=Y,signature=signature,X.test=X)

pred2=prediction(X=X,Y=Y,signature=signature,X.test=X,CI=TRUE)
lapply(pred2$out.CI$CI$'comp.1',head)
lapply(pred2$out.CI$CI$'comp.2',head)
lapply(pred2$out.CI$CI$'comp.3',head)


## End(Not run)
```

spls.hybrid — *spls.hybrid, midway between PLS and sPLS*

Description

Performs a constraint sPLS on the first PLS-components and a sPLS on the last components

Usage

```
spls.hybrid(X,
            Y,
            ncomp = 2,
            mode = c("regression", "canonical", "invariant", "classic"),
            max.iter = 500,
            tol = 1e-06,
            keepX.constraint,
            keepY.constraint,
            keepX,
            keepY,
            near.zero.var = FALSE)
```

Arguments

`X`	numeric matrix of predictors. `NA`s are allowed.
`Y`	numeric vector or matrix of responses (for multi-response models). `NA`s are allowed.
`ncomp`	the number of components to include in the model (see Details). Default is 2.
`mode`	character string. What type of algorithm to use. one of `"regression"`, `"canonical"`, `"invariant"` or `"classic"`. See Details.
`max.iter`	integer, the maximum number of iterations.
`tol`	a positive real, the tolerance used in the iterative algorithm.
`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 X variables kept in the model on the last components.
`keepY`	number of Y variables kept in the model on the last components.
`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.

Details

The `spls.hybrid` function allows you to compute a constraint spls on the first components and a spls on the last components. Note that the only condition on `keepX.constraint` and `keepX` is that the sum of both length is ncomp; likewise for the ones relative to Y.

Value

A 'spls.hybrid' object is returned. The object is 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.constraint`	A list of length `ncomp` containing which variables of X are to be kept on each component.
`keepY.constraint`	A list of length `ncomp` containing which variables of Y are to be kept on each component.
`mat.c`	matrix of coefficients to be used internally by `predict`.
`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, for `X` and `Y`.
`coeff`	A list of means.X, sigma.X, means.Y and sigma.Y. Means and variances for the variables of `X` and the columns of `Y.mat`.

References

Rohart *et al.* (2016). A Molecular Classification of Human Mesenchymal Stromal Cells. PeerJ, DOI 10.7717/peerj.1845

Examples

```
## Not run:
data(MSC)
X=MSC$X
Y=MSC$Y

Y.mat=unmap(Y)

n=nrow(X)
p=ncol(X)


keepX.constraint=list(sample(1:p,7),sample(1:p,15))
fit=spls.hybrid(X,Y=Y.mat,ncomp=4,keepX.constraint=keepX.constraint,keepX=c(5,10))

plotIndiv(fit,ind.names=FALSE)

## End(Not run)
```

`variable.selection` *Tune the number of variables on each component*

Description

Performs a testing procedure to choose the number of variable on each component

Usage

```
variable.selection(object,ncomp,alpha,limit,showProgress=TRUE)
```

Arguments

`object`	a 'bootsPLS' object', as obtained from `bootsPLS`
`ncomp`	How many component?
`alpha`	Level of the test
`limit`	Vector of maximal number of genes to include on each component
`showProgress`	Logical. If TRUE, shows the progress of the algorithm.

Details

The testing procedure ranks the variables by decreasing frequency in `object$frequency`, for each component. Random subsamplings are constructed, a `spls.hybrid` is fitted on the internal learning set and a prediction is made on the internal test set. The testing procedure evaluates the gain in classification accuracy over the random subsamplings when a new variable is added from a decreasing frequency. This is done by on-sided t-test of level `alpha`. See the reference below for more details on the multiple testing procedure.

Value

A 'variable.selection' object is returned for which `plot` is available.

`pval`	pvalue obtained from the testing procedure.
`opt`	Number of components chosen by the procedure
`signature`	Variables chosen on each of the `ncomp` components
`object`	input 'bootsPLS' object
`alpha`	input level of the test

References

Rohart *et al.* (2016). A Molecular Classification of Human Mesenchymal Stromal Cells. PeerJ, DOI 10.7717/peerj.1845

See Also

`spls.hybrid`, `plot`, `fit.model`, `prediction`

Examples

```
## Not run:
data(MSC)
X=MSC$X
Y=MSC$Y
dim(X)
table(Y)

boot=bootsPLS(X=X,Y=Y,ncomp=3,many=5,kCV=5)
out=variable.selection(boot,ncomp=2)

out2=variable.selection(boot,ncomp=2,limit=c(40,40))


## End(Not run)
```

Index