Package 'plsgenomics'

October 14, 2022

Version 1.5-2

Date 2018-08-23

Title PLS Analyses for Genomics

Author Anne-Laure Boulesteix <boulesteix@ibe.med.uni-muenchen.de>,
Ghislain Durif <gd.dev@libertymail.net>,
Sophie Lambert-Lacroix
<sophie.lambert-lacroix@univ-grenoble-alpes.fr>, Julie Peyre
<Julie.Peyre@univ-grenoble-alpes.fr>, and Korbinian Strimmer
<k.strimmer@imperial.ac.uk>.

Maintainer Ghislain Durif <gd.dev@libertymail.net>

Imports MASS, boot, parallel, reshape2, plyr, fields, RhpcBLASctl

Depends R (>= 3.0)

Suggests

Encoding latin1

Description Routines for PLS-based genomic analyses,
implementing PLS methods for classification with
microarray data and prediction of transcription factor
activities from combined ChIP-chip analysis. The >=1.2-1
versions include two new classification methods for microarray
data: GSIM and Ridge PLS. The >=1.3 versions includes a
new classification method combining variable selection and
compression in logistic regression context: logit-SPLS; and
an adaptive version of the sparse PLS.

License GPL (>= 2)

URL https://CRAN.R-project.org/package=plsgenomics

Repository CRAN

Date/Publication 2018-08-24 08:30:03 UTC

RoxygenNote 6.0.1

NeedsCompilation no

R topics documented:

`Colon` *Gene expression data from Alon et al. (1999)*

Description

Gene expression data (2000 genes for 62 samples) from the microarray experiments of Colon tissue samples of Alon et al. (1999).

Usage

```
data(Colon)
```

Details

This data set contains 62 samples with 2000 genes: 40 tumor tissues, coded 2 and 22 normal tissues, coded 1.

Value

A list with the following elements:

`X`	a (62 x 2000) matrix giving the expression levels of 2000 genes for the 62 Colon tissue samples. Each row corresponds to a patient, each column to a gene.
`Y`	a numeric vector of length 62 giving the type of tissue sample (tumor or normal).
`gene.names`	a vector containing the names of the 2000 genes for the gene expression matrix `X`.

Source

The data are described in Alon et al. (1999) and can be freely downloaded from http://microarray.princeton.edu/oncology/affydata/index.html.

References

Alon, U. and Barkai, N. and Notterman, D.A. and Gish, K. and Ybarra, S. and Mack, D. and Levine, A.J. (1999). Broad patterns of gene expression revealed by clustering analysis of tumor and normal colon tissues probed by oligonucleotide arrays, Proc. Natl. Acad. Sci. USA,**96**(12), 6745–6750.

Examples

```
# load plsgenomics library
library(plsgenomics)

# load data set
data(Colon)

# how many samples and how many genes ?
dim(Colon$X)

# how many samples of class 1 and 2 respectively ?
sum(Colon$Y==1)
sum(Colon$Y==2)
```

Ecoli *Ecoli gene expression and connectivity data from Kao et al. (2003)*

Description

Gene expression data obtained during Escherichia coli carbon source transition and connectivity data from the RegulonDB data base (Salgado et al., 2001). The experiments and data sets are described in Kao et al. (2003).

Usage

```
data(Ecoli)
```

Value

A list with the following components:

`CONNECdata`	a (100 x 16) matrix containing the connectivity data for 100 genes and 16 regulators. The data are coded as 1 (positive interaction), 0 (no interaction) and -1 (negative interaction).
`GEdata`	a (100 x 23) matrix containing gene expression data for 100 genes and 23 samples corresponding to differents times during carbon source transition.
`timepoint`	a numeric vector of length 23 containing the time points (in hours) for the 23 samples.

Source

The data are described in Kao et al. (2004) and can be freely downloaded from http://www.seas.ucla.edu/~liaoj/downloads.html.

References

K. Kao, Y.-L. Yang, R. Boscolo, C. Sabatti, V. Roychowdhury and J. C. Liao (2004). Transcriptome-based determination of multiple transcription regulator activities in Escherichia coli by using network component analysis, PNAS **101**, 641–646.

H. Salgado, A. Santos-Zavaleta, S. Gama-Castro, D. Millan-Zarate, E. Diaz-Peredo, F. Sanchez-Solano, E. Perez-Rueda, C. Bonavides-Martinez and J. Collado-Vides (2001). RegulonDB (version 3.2): transcriptional regulation and operon organization in Escherichia coli K-12, Nucleic Acids Research **29**, 72–74.

Examples

```
# load plsgenomics library
library(plsgenomics)

# load data set
data(Ecoli)
```

```
# how many genes and how many transcription factors ?
dim(Ecoli$CONNECdata)
```

---

gsim *GSIM for binary data*

Description

The function gsim performs prediction using Lambert-Lacroix and Peyre's GSIM algorithm.

Usage

```
gsim(Xtrain, Ytrain, Xtest=NULL, Lambda, hA, hB=NULL, NbIterMax=50)
```

Arguments

Xtrain	a (ntrain x p) data matrix of predictors. Xtrain must be a matrix. Each row corresponds to an observation and each column to a predictor variable.
Ytrain	a ntrain vector of responses. Ytrain must be a vector. Ytrain is a {1,2}-valued vector and contains the response variable for each observation.
Xtest	a (ntest x p) matrix containing the predictors for the test data set. Xtest may also be a vector of length p (corresponding to only one test observation). If Xtest is not equal to NULL, then the prediction step is made for these new predictor variables.
Lambda	a positive real value. Lambda is the ridge regularization parameter.
hA	a strictly positive real value. hA is the bandwidth for GSIM step A.
hB	a strictly positive real value. hB is the bandwidth for GSIM step B. if hB is equal to NULL, then hB value is chosen using a plug-in method.
NbIterMax	a positive integer. NbIterMax is the maximal number of iterations in the Newton-Rapson parts.

Details

The columns of the data matrices Xtrain and Xtest may not be standardized, since standardizing is performed by the function gsim as a preliminary step before the algorithm is run.

The procedure described in Lambert-Lacroix and Peyre (2005) is used to estimate the projection direction beta. When Xtest is not equal to NULL, the procedure predicts the labels for these new predictor variables.

Value

A list with the following components:

`Ytest`	the ntest vector containing the predicted labels for the observations from `Xtest`.
`beta`	the p vector giving the projection direction estimated.
`hB`	the value of hB used in step B of GSIM (value given by the user or estimated by plug-in if the argument value was equal to NULL)
`DeletedCol`	the vector containing the column number of `Xtrain` when the variance of the corresponding predictor variable is null. Otherwise `DeletedCol`=NULL
`Cvg`	the 0-1 value indicating convergence of the algorithm (1 for convergence, 0 otherwise).

Author(s)

Sophie Lambert-Lacroix (http://membres-timc.imag.fr/Sophie.Lambert/) and Julie Peyre (http://www-lmc.imag.fr/lmc-sms/Julie.Peyre/).

References

S. Lambert-Lacroix, J. Peyre . (2006) Local likelyhood regression in generalized linear single-index models with applications to microarrays data. Computational Statistics and Data Analysis, vol 51, n 3, 2091-2113.

See Also

`gsim.cv`, `mgsim`, `mgsim.cv`.

Examples

```
# load plsgenomics library
library(plsgenomics)

# load Colon data
data(Colon)
IndexLearn <- c(sample(which(Colon$Y==2),12),sample(which(Colon$Y==1),8))

Xtrain <- Colon$X[IndexLearn,]
Ytrain <- Colon$Y[IndexLearn]
Xtest <- Colon$X[-IndexLearn,]

# preprocess data
resP <- preprocess(Xtrain= Xtrain, Xtest=Xtest,Threshold = c(100,16000),Filtering=c(5,500),
log10.scale=TRUE,row.stand=TRUE)

# perform prediction by GSIM
res <- gsim(Xtrain=resP$pXtrain,Ytrain= Ytrain,Xtest=resP$pXtest,Lambda=10,hA=50,hB=NULL)

res$Cvg
sum(res$Ytest!=Colon$Y[-IndexLearn])
```

gsim.cv *Determination of the ridge regularization parameter and the bandwidth to be used for classification with GSIM for binary data*

Description

The function gsim.cv determines the best ridge regularization parameter and bandwidth to be used for classification with GSIM as described in Lambert-Lacroix and Peyre (2005).

Usage

```
gsim.cv(Xtrain, Ytrain,LambdaRange,hARange,hB=NULL, NbIterMax=50)
```

Arguments

Xtrain	a (ntrain x p) data matrix of predictors. Xtrain must be a matrix. Each row corresponds to an observation and each column to a predictor variable.
Ytrain	a ntrain vector of responses. Ytrain must be a vector. Ytrain is a {1,2}-valued vector and contains the response variable for each observation.
LambdaRange	the vector of positive real value from which the best ridge regularization parameter has to be chosen by cross-validation.
hARange	the vector of strictly positive real value from which the best bandwidth has to be chosen by cross-validation for GSIM step A.
hB	a strictly positive real value. hB is the bandwidth for GSIM step B. if hB is equal to NULL, then hB value is chosen using a plug-in method.
NbIterMax	a positive integer. NbIterMax is the maximal number of iterations in the Newton-Rapson parts.

Details

The cross-validation procedure described in Lambert-Lacroix and Peyre (2005) is used to determine the best ridge regularization parameter and bandwidth to be used for classification with GSIM for binary data (for categorical data see mgsim and mgsim.cv). At each cross-validation run, Xtrain is split into a pseudo training set (ntrain - 1 samples) and a pseudo test set (1 sample) and the classification error rate is determined for each value of ridge regularization parameter and bandwidth. Finally, the function gsim.cv returns the values of the ridge regularization parameter and bandwidth for which the mean classification error rate is minimal.

Value

A list with the following components:

Lambda	the optimal regularization parameter.
hA	the optimal bandwidth parameter.

Author(s)

Sophie Lambert-Lacroix (http://membres-timc.imag.fr/Sophie.Lambert/) and Julie Peyre (http://www-lmc.imag.fr/lmc-sms/Julie.Peyre/).

References

S. Lambert-Lacroix, J. Peyre . (2006) Local likelyhood regression in generalized linear single-index models with applications to microarrays data. Computational Statistics and Data Analysis, vol 51, n 3, 2091-2113.

See Also

mgsim, gsim, gsim.cv.

Examples

```
## Not run:
## between 5~15 seconds
# load plsgenomics library
library(plsgenomics)

# load Colon data
data(Colon)
IndexLearn <- c(sample(which(Colon$Y==2),12),sample(which(Colon$Y==1),8))

Xtrain <- Colon$X[IndexLearn,]
Ytrain <- Colon$Y[IndexLearn]
Xtest <- Colon$X[-IndexLearn,]

# preprocess data
resP <- preprocess(Xtrain= Xtrain, Xtest=Xtest,Threshold = c(100,16000),Filtering=c(5,500),
log10.scale=TRUE,row.stand=TRUE)

# Determine optimum h and lambda
hl <- gsim.cv(Xtrain=resP$pXtrain,Ytrain=Ytrain,hARange=c(7,20),LambdaRange=c(0.1,1),hB=NULL)

# perform prediction by GSIM
res <- gsim(Xtrain=resP$pXtrain,Ytrain=Ytrain,Xtest=resP$pXtest,Lambda=hl$Lambda,hA=hl$hA,hB=NULL)
res$Cvg
sum(res$Ytest!=Colon$Y[-IndexLearn])

## End(Not run)
```

---

leukemia *Gene expression data from Golub et al. (1999)*

Description

Gene expression data (3051 genes and 38 tumor mRNA samples) from the leukemia microarray study of Golub et al. (1999).

Usage

```
data(leukemia)
```

Value

A list with the following elements:

`X`	a (38 x 3051) matrix giving the expression levels of 3051 genes for 38 leukemia patients. Each row corresponds to a patient, each column to a gene.
`Y`	a numeric vector of length 38 giving the cancer class of each patient.
`gene.names`	a matrix containing the names of the 3051 genes for the gene expression matrix `X`. The three columns correspond to the gene 'index', 'ID', and 'Name', respectively.

Source

The dataset was taken from the R package multtest. The data are described in Golub et al. (1999) and can be freely downloaded from http://www.broadinstitute.org/cgi-bin/cancer/publications/pub_paper.cgi?paper_id=43.

References

S. Dudoit, J. Fridlyand and T. P. Speed (2002). Comparison of discrimination methods for the classification of tumors using gene expression data, Journal of the American Statistical Association **97**, 77–87.

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

Examples

```
# load plsgenomics library
library(plsgenomics)

# load data set
data(leukemia)

# how many samples and how many genes ?
dim(leukemia$X)

# how many samples of class 1 and 2, respectively ?
sum(leukemia$Y==1)
sum(leukemia$Y==2)
```

logit.spls *Classification procedure for binary response based on a logistic model, solved by a combination of the Ridge Iteratively Reweighted Least Squares (RIRLS) algorithm and the Adaptive Sparse PLS (SPLS) regression*

Description

The function `logit.spls` performs compression and variable selection in the context of binary classification (with possible prediction) using Durif et al. (2017) algorithm based on Ridge IRLS and sparse PLS.

Usage

```
logit.spls(Xtrain, Ytrain, lambda.ridge, lambda.l1, ncomp, Xtest = NULL,
  adapt = TRUE, maxIter = 100, svd.decompose = TRUE, center.X = TRUE,
  scale.X = FALSE, weighted.center = TRUE)
```

Arguments

`Xtrain`	a (ntrain x p) data matrix of predictor values. `Xtrain` must be a matrix. Each row corresponds to an observation and each column to a predictor variable.
`Ytrain`	a (ntrain) vector of (continuous) responses. `Ytrain` must be a vector or a one column matrix, and contains the response variable for each observation. `Ytrain` should take values in {0,1}.
`lambda.ridge`	a positive real value. `lambda.ridge` is the Ridge regularization parameter for the RIRLS algorithm (see details).
`lambda.l1`	a positive real value, in [0,1]. `lambda.l1` is the sparse penalty parameter for the dimension reduction step by sparse PLS (see details).
`ncomp`	a positive integer. `ncomp` is the number of PLS components. If `ncomp=0`,then the Ridge regression is performed without any dimension reduction (no SPLS step).
`Xtest`	a (ntest x p) matrix containing the predictor values for the test data set. `Xtest` may also be a vector of length p (corresponding to only one test observation). Default value is NULL, meaning that no prediction is performed.
`adapt`	a boolean value, indicating whether the sparse PLS selection step sould be adaptive or not (see details).
`maxIter`	a positive integer. `maxIter` is the maximal number of iterations in the Newton-Raphson parts in the RIRLS algorithm (see details).
`svd.decompose`	a boolean parameter. `svd.decompose` indicates wether or not the predictor matrix `Xtrain` should be decomposed by SVD (singular values decomposition) for the RIRLS step (see details).
`center.X`	a boolean value indicating whether the data matrices `Xtrain` and `Xtest` (if provided) should be centered or not.

`scale.X` a boolean value indicating whether the data matrices `Xtrain` and `Xtest` (if provided) should be scaled or not (`scale.X=TRUE` implies `center.X=TRUE`) in the spls step.

`weighted.center` a boolean value indicating whether the centering should take into account the weighted l2 metric or not in the SPLS step.

Details

The columns of the data matrices `Xtrain` and `Xtest` may not be standardized, since standardizing can be performed by the function `logit.spls` as a preliminary step.

The procedure described in Durif et al. (2017) is used to compute latent sparse components that are used in a logistic regression model. In addition, when a matrix `Xtest` is supplied, the procedure predicts the response associated to these new values of the predictors.

Value

An object of class `logit.spls` with the following attributes

`Coefficients` the (p+1) vector containing the linear coefficients associated to the predictors and intercept in the logistic model explaining the response Y.

`hatY` the (ntrain) vector containing the estimated response value on the train set `Xtrain`.

`hatYtest` the (ntest) vector containing the predicted labels for the observations from `Xtest` (if provided).

`DeletedCol` the vector containing the indexes of columns with null variance in `Xtrain` that were skipped in the procedure.

`A` the active set of predictors selected by the procedures. `A` is a subset of 1:p.

`Anames` Vector of selected predictor names, i.e. the names of the columns from `Xtrain` that are in `A`.

`converged` a {0,1} value indicating whether the RIRLS algorithm did converge in less than `maxIter` iterations or not.

`X.score` a (n x ncomp) matrix being the observations coordinates or scores in the new component basis produced by the SPLS step (sparse PLS). Each column t.k of `X.score` is a SPLS component.

`X.weight` a (p x ncomp) matrix being the coefficients of predictors in each components produced by sparse PLS. Each column w.k of `X.weight` verifies t.k = Xtrain x w.k (as a matrix product).

`Xtrain` the design matrix.

`sXtrain` the scaled predictor matrix.

`Ytrain` the response observations.

`sPseudoVar` the scaled pseudo-response produced by the RIRLS algorithm.

`lambda.ridge` the Ridge hyper-parameter used to fit the model.

`lambda.l1` the sparse hyper-parameter used to fit the model.

`ncomp` the number of components used to fit the model.

V	the (ntrain x ntrain) matrix used to weight the metric in the sparse PLS step. V is the inverse of the covariance matrix of the pseudo-response produced by the RIRLS step.
proba	the (ntrain) vector of estimated probabilities for the observations in code Xtrain, that are used to estimate the hatY labels.
proba.test	the (ntest) vector of predicted probabilities for the new observations in Xtest, that are used to predict the hatYtest labels.

Author(s)

Ghislain Durif (http://thoth.inrialpes.fr/people/gdurif/).

References

Durif G., Modolo L., Michaelsson J., Mold J. E., Lambert-Lacroix S., Picard F. (2017). High Dimensional Classification with combined Adaptive Sparse PLS and Logistic Regression, (in prep), available on (http://arxiv.org/abs/1502.05933).

See Also

spls, logit.spls.cv

Examples

```
## Not run:
### load plsgenomics library
library(plsgenomics)

### generating data
n <- 100
p <- 100
sample1 <- sample.bin(n=n, p=p, kstar=20, lstar=2,
                      beta.min=0.25, beta.max=0.75,
                      mean.H=0.2, sigma.H=10, sigma.F=5)
X <- sample1$X
Y <- sample1$Y

### splitting between learning and testing set
index.train <- sort(sample(1:n, size=round(0.7*n)))
index.test <- (1:n)[-index.train]

Xtrain <- X[index.train,]
Ytrain <- Y[index.train,]
Xtest <- X[index.test,]
Ytest <- Y[index.test,]

### fitting the model, and predicting new observations
model1 <- logit.spls(Xtrain=Xtrain, Ytrain=Ytrain, lambda.ridge=2,
                     lambda.l1=0.5, ncomp=2, Xtest=Xtest, adapt=TRUE,
                     maxIter=100, svd.decompose=TRUE)
```

```
str(model1)

### prediction error rate
sum(model1$hatYtest!=Ytest) / length(index.test)

## End(Not run)
```

`logit.spls.cv` *Cross-validation procedure to calibrate the parameters (ncomp, lambda.l1, lambda.ridge) for the LOGIT-SPLS method*

Description

The function `logit.spls.cv` chooses the optimal values for the hyper-parameter of the `logit.spls` procedure, by minimizing the averaged error of prediction over the hyper-parameter grid, using Durif et al. (2017) LOGIT-SPLS algorithm.

Usage

```
logit.spls.cv(X, Y, lambda.ridge.range, lambda.l1.range, ncomp.range,
  adapt = TRUE, maxIter = 100, svd.decompose = TRUE,
  return.grid = FALSE, ncores = 1, nfolds = 10, nrun = 1,
  center.X = TRUE, scale.X = FALSE, weighted.center = TRUE, seed = NULL,
  verbose = TRUE)
```

Arguments

`X`	a (n x p) data matrix of predictors. `X` must be a matrix. Each row corresponds to an observation and each column to a predictor variable.
`Y`	a (n) vector of (continuous) responses. `Y` must be a vector or a one column matrix. It contains the response variable for each observation. `Y` should take values in {0,1}.
`lambda.ridge.range`	a vector of positive real values. `lambda.ridge` is the Ridge regularization parameter for the RIRLS algorithm (see details), the optimal value will be chosen among `lambda.ridge.range`.
`lambda.l1.range`	a vecor of positive real values, in [0,1]. `lambda.l1` is the sparse penalty parameter for the dimension reduction step by sparse PLS (see details), the optimal value will be chosen among `lambda.l1.range`.
`ncomp.range`	a vector of positive integers. `ncomp` is the number of PLS components. The optimal value will be chosen among `ncomp.range`.
`adapt`	a boolean value, indicating whether the sparse PLS selection step sould be adaptive or not (see details).

maxIter	a positive integer, the maximal number of iterations in the RIRLS algorithm (see details).
svd.decompose	a boolean parameter. svd.decompose indicates wether or not the predictor matrix Xtrain should be decomposed by SVD (singular values decomposition) for the RIRLS step (see details).
return.grid	a boolean values indicating whether the grid of hyper-parameters values with corresponding mean prediction error rate over the folds should be returned or not.
ncores	a positve integer, indicating the number of cores that the cross-validation is allowed to use for parallel computation (see details).
nfolds	a positive integer indicating the number of folds in the K-folds cross-validation procedure, nfolds=n corresponds to the leave-one-out cross-validation, default is 10.
nrun	a positive integer indicating how many times the K-folds cross- validation procedure should be repeated, default is 1.
center.X	a boolean value indicating whether the data matrices Xtrain and Xtest (if provided) should be centered or not.
scale.X	a boolean value indicating whether the data matrices Xtrain and Xtest (if provided) should be scaled or not (scale.X=TRUE implies center.X=TRUE) in the spls step.
weighted.center	a boolean value indicating whether the centering should take into account the weighted l2 metric or not in the SPLS step.
seed	a positive integer value (default is NULL). If non NULL, the seed for pseudo-random number generation is set accordingly.
verbose	a boolean parameter indicating the verbosity.

Details

The columns of the data matrices X may not be standardized, since standardizing is performed by the function logit.spls.cv as a preliminary step.

The procedure is described in Durif et al. (2017). The K-fold cross-validation can be summarize as follow: the train set is partitioned into K folds, for each value of hyper-parameters the model is fit K times, using each fold to compute the prediction error rate, and fitting the model on the remaining observations. The cross-validation procedure returns the optimal hyper-parameters values, meaning the one that minimize the averaged error of prediction averaged over all the folds.

This procedures uses mclapply from the parallel package, available on GNU/Linux and MacOS. Users of Microsoft Windows can refer to the README file in the source to be able to use a mclapply type function.

Value

An object of class logit.spls with the following attributes

lambda.ridge.opt	the optimal value in lambda.ridge.range.

lambda.l1.opt	the optimal value in lambda.l1.range.
ncomp.opt	the optimal value in ncomp.range.
conv.per	the overall percentage of models that converge during the cross-validation procedure.
cv.grid	the grid of hyper-parameters and corresponding prediction error rate averaged over the folds. cv.grid is NULL if return.grid is set to FALSE.

Author(s)

Ghislain Durif (http://thoth.inrialpes.fr/people/gdurif/).

References

Durif G., Modolo L., Michaelsson J., Mold J. E., Lambert-Lacroix S., Picard F. (2017). High Dimensional Classification with combined Adaptive Sparse PLS and Logistic Regression, (in prep), available on (http://arxiv.org/abs/1502.05933).

See Also

logit.spls, logit.spls.stab

Examples

```
## Not run:
### load plsgenomics library
library(plsgenomics)

### generating data
n <- 100
p <- 100
sample1 <- sample.bin(n=n, p=p, kstar=10, lstar=2,
                      beta.min=0.25, beta.max=0.75, mean.H=0.2,
                      sigma.H=10, sigma.F=5)

X <- sample1$X
Y <- sample1$Y

### hyper-parameters values to test
lambda.l1.range <- seq(0.05,0.95,by=0.1) # between 0 and 1
ncomp.range <- 1:10
# log-linear range between 0.01 a,d 1000 for lambda.ridge.range
logspace <- function( d1, d2, n) exp(log(10)*seq(d1, d2, length.out=n))
lambda.ridge.range <- signif(logspace(d1 <- -2, d2 <- 3, n=21), digits=3)

### tuning the hyper-parameters
cv1 <- logit.spls.cv(X=X, Y=Y, lambda.ridge.range=lambda.ridge.range,
                     lambda.l1.range=lambda.l1.range,
                     ncomp.range=ncomp.range,
                     adapt=TRUE, maxIter=100, svd.decompose=TRUE,
                     return.grid=TRUE, ncores=1, nfolds=10)
```

```
str(cv1)

## End(Not run)
```

`logit.spls.stab` *Stability selection procedure to estimate probabilities of selection of covariates for the LOGIT-SPLS method*

Description

The function `logit.spls.stab` train a logit-spls model for each candidate values (`ncomp`, `lambda.l1`, `lambda.ridge`) of hyper-parameters on multiple sub-samplings in the data. The stability selection procedure selects the covariates that are selected by most of the models among the grid of hyper-parameters, following the procedure described in Durif et al. (2017). Candidates values for `ncomp`, `lambda.l1` and `lambda.l2` are respectively given by the input arguments `ncomp.range`, `lambda.l1.range` and `lambda.l2.range`.

Usage

```
logit.spls.stab(X, Y, lambda.ridge.range, lambda.l1.range, ncomp.range,
  adapt = TRUE, maxIter = 100, svd.decompose = TRUE, ncores = 1,
  nresamp = 100, center.X = TRUE, scale.X = FALSE,
  weighted.center = TRUE, seed = NULL, verbose = TRUE)
```

Arguments

`X`	a (n x p) data matrix of predictors. `X` must be a matrix. Each row corresponds to an observation and each column to a predictor variable.
`Y`	a (n) vector of (continuous) responses. `Y` must be a vector or a one column matrix. It contains the response variable for each observation. `Y` should take values in {0,1}.
`lambda.ridge.range`	a vector of positive real values. `lambda.ridge` is the Ridge regularization parameter for the RIRLS algorithm (see details), the optimal value will be chosen among `lambda.ridge.range`.
`lambda.l1.range`	a vecor of positive real values, in [0,1]. `lambda.l1` is the sparse penalty parameter for the dimension reduction step by sparse PLS (see details), the optimal value will be chosen among `lambda.l1.range`.
`ncomp.range`	a vector of positive integers. `ncomp` is the number of PLS components. The optimal value will be chosen among `ncomp.range`.
`adapt`	a boolean value, indicating whether the sparse PLS selection step sould be adaptive or not (see details).
`maxIter`	a positive integer, the maximal number of iterations in the RIRLS algorithm (see details).

svd.decompose	a boolean parameter. svd.decompose indicates wether or not the predictor matrix Xtrain should be decomposed by SVD (singular values decomposition) for the RIRLS step (see details).
ncores	a positve integer, indicating the number of cores that the cross-validation is allowed to use for parallel computation (see details).
nresamp	number of resamplings of the data to estimate the probility of selection for each covariate, default is 100.
center.X	a boolean value indicating whether the data matrices Xtrain and Xtest (if provided) should be centered or not.
scale.X	a boolean value indicating whether the data matrices Xtrain and Xtest (if provided) should be scaled or not (scale.X=TRUE implies center.X=TRUE) in the spls step.
weighted.center	a boolean value indicating whether the centering should take into account the weighted l2 metric or not in the SPLS step.
seed	a positive integer value (default is NULL). If non NULL, the seed for pseudo-random number generation is set accordingly.
verbose	a boolean parameter indicating the verbosity.

Details

The columns of the data matrices X may not be standardized, since standardizing is performed by the function logit.spls.stab as a preliminary step.

The procedure is described in Durif et al. (2017). The stability selection procedure can be summarize as follow (c.f. Meinshausen and Buhlmann, 2010).

(i) For each candidate values (ncomp, lambda.l1, lambda.ridge) of hyper-parameters, a logit-SPLS is trained on nresamp resamplings of the data. Then, for each triplet (ncomp, lambda.l1, lambda.ridge), the probability that a covariate (i.e. a column in X) is selected is computed among the resamplings.

(ii) Eventually, the set of "stable selected" variables corresponds to the set of covariates that were selected by most of the training among the grid of hyper-parameters candidate values.

This function achieves the first step (i) of the stability selection procedure. The second step (ii) is achieved by the function stability.selection.

This procedures uses mclapply from the parallel package, available on GNU/Linux and MacOS. Users of Microsoft Windows can refer to the README file in the source to be able to use a mclapply type function.

Value

An object with the following attributes

q.Lambda	A table with values of q.Lambda (c.f. Durif et al. (2017) for the notation), being the averaged number of covariates selected among the entire grid of hyper-parameters candidates values, for increasing size of hyper-parameter grid.
probs.lambda	A table with estimated probability of selection for each covariates depending on the candidates values for hyper-parameters.
p	An integer values indicating the number of covariates in the model.

Author(s)

Ghislain Durif (http://thoth.inrialpes.fr/people/gdurif/).

References

Durif G., Modolo L., Michaelsson J., Mold J. E., Lambert-Lacroix S., Picard F. (2017). High Dimensional Classification with combined Adaptive Sparse PLS and Logistic Regression, (in prep), available on (http://arxiv.org/abs/1502.05933).

Meinshausen, N., Buhlmann P. (2010). Stability Selection. Journal of the Royal Statistical Society: Series B (Statistical Methodology) 72, no. 4, 417-473.

See Also

logit.spls, stability.selection, stability.selection.heatmap

Examples

```
## Not run:
### load plsgenomics library
library(plsgenomics)

### generating data
n <- 100
p <- 100
sample1 <- sample.bin(n=n, p=p, kstar=10, lstar=2,
                      beta.min=0.25, beta.max=0.75, mean.H=0.2,
                      sigma.H=10, sigma.F=5)

X <- sample1$X
Y <- sample1$Y

### pertinent covariates id
sample1$sel

### hyper-parameters values to test
lambda.l1.range <- seq(0.05,0.95,by=0.1) # between 0 and 1
ncomp.range <- 1:10
# log-linear range between 0.01 a,d 1000 for lambda.ridge.range
logspace <- function( d1, d2, n) exp(log(10)*seq(d1, d2, length.out=n))
lambda.ridge.range <- signif(logspace(d1 <- -2, d2 <- 3, n=21), digits=3)

### tuning the hyper-parameters
stab1 <- logit.spls.stab(X=X, Y=Y, lambda.ridge.range=lambda.ridge.range,
                         lambda.l1.range=lambda.l1.range,
                         ncomp.range=ncomp.range,
                         adapt=TRUE, maxIter=100, svd.decompose=TRUE,
                         ncores=1, nresamp=100)

str(stab1)

### heatmap of estimated probabilities
```

```
stability.selection.heatmap(stab1)

### selected covariates
stability.selection(stab1, piThreshold=0.6, rhoError=10)

## End(Not run)
```

`matrix.heatmap` *Heatmap visualization for matrix*

Description

Visualization of matrix entries in heatmap format, the color scale depends on the numerical values.

Usage

```
matrix.heatmap(mat, ...)
```

Arguments

`mat`	the matrix to visualize
`...`	any argument that could be pass to the functions `image.plot` or `image`.

Details

The function `matrix.heatmap` is a wrapper for the function `image.plot` from the 'fields' package.

Value

No return, just plot the heatmap in the current graphic window.

Author(s)

Ghislain Durif (http://thoth.inrialpes.fr/people/gdurif/).

See Also

`logit.spls`, `stability.selection`, `stability.selection.heatmap`

Examples

```
### load plsgenomics library
library(plsgenomics)

### generate a matrix
A = matrix(runif(10*10), ncol=10)

### heatmap of estimated probabilities
matrix.heatmap(A)
```

mgsim *GSIM for categorical data*

Description

The function mgsim performs prediction using Lambert-Lacroix and Peyre's MGSIM algorithm.

Usage

```
mgsim(Ytrain,Xtrain,Lambda,h,Xtest=NULL,NbIterMax=50)
```

Arguments

Xtrain	a (ntrain x p) data matrix of predictors. Xtrain must be a matrix. Each row corresponds to an observation and each column to a predictor variable.
Ytrain	a ntrain vector of responses. Ytrain must be a vector. Ytrain is a {1,...,c+1}-valued vector and contains the response variable for each observation. c+1 is the number of classes.
Xtest	a (ntest x p) matrix containing the predictors for the test data set. Xtest may also be a vector of length p (corresponding to only one test observation). If Xtest is not equal to NULL, then the prediction step is made for these new predictor variables.
Lambda	a positive real value. Lambda is the ridge regularization parameter.
h	a strictly positive real value. h is the bandwidth for GSIM step A.
NbIterMax	a positive integer. NbIterMax is the maximal number of iterations in the Newton-Rapson parts.

Details

The columns of the data matrices Xtrain and Xtest may not be standardized, since standardizing is performed by the function mgsim as a preliminary step before the algorithm is run.

The procedure described in Lambert-Lacroix and Peyre (2005) is used to estimate the c projection directions and the coefficients of the parametric fit obtained after projecting predictor variables onto the estimated directions. When Xtest is not equal to NULL, the procedure predicts the labels for these new predictor variables.

Value

A list with the following components:

Ytest	the ntest vector containing the predicted labels for the observations from Xtest.
beta	the (p x c) matrix containing the c estimated projection directions.
Coefficients	the (2 x c) matrix containing the coefficients of the parametric fit obtained after projecting predictor variables onto these estimated directions.

DeletedCol	the vector containing the column number of Xtrain when the variance of the corresponding predictor variable is null. Otherwise DeletedCol=NULL
Cvg	the 0-1 value indicating convergence of the algorithm (1 for convergence, 0 otherwise).

Author(s)

Sophie Lambert-Lacroix (http://membres-timc.imag.fr/Sophie.Lambert/) and Julie Peyre (http://www-lmc.imag.fr/lmc-sms/Julie.Peyre/).

References

S. Lambert-Lacroix, J. Peyre . (2006) Local likelyhood regression in generalized linear single-index models with applications to microarrays data. Computational Statistics and Data Analysis, vol 51, n 3, 2091-2113.

See Also

mgsim.cv, gsim, gsim.cv.

Examples

```
# load plsgenomics library
library(plsgenomics)

# load SRBCT data
data(SRBCT)
IndexLearn <- c(sample(which(SRBCT$Y==1),10),sample(which(SRBCT$Y==2),4),
sample(which(SRBCT$Y==3),7),sample(which(SRBCT$Y==4),9))

# perform prediction by MGSIM
res <- mgsim(Ytrain=SRBCT$Y[IndexLearn],Xtrain=SRBCT$X[IndexLearn,],Lambda=0.001,h=19,
Xtest=SRBCT$X[-IndexLearn,])
res$Cvg
sum(res$Ytest!=SRBCT$Y[-IndexLearn])

# prediction for another sample
Xnew <- SRBCT$X[83,]
# projection of Xnew onto the c estimated direction
Xproj <- Xnew %*% res$beta
# Compute the linear predictor for each classes expect class 1
eta <- diag(cbind(rep(1,3),t(Xproj)) %*% res$Coefficients)
Ypred <- which.max(c(0,eta))
Ypred
SRBCT$Y[83]
```

`mgsim.cv` *Determination of the ridge regularization parameter and the bandwidth to be used for classification with GSIM for categorical data*

Description

The function `mgsim.cv` determines the best ridge regularization parameter and bandwidth to be used for classification with MGSIM as described in Lambert-Lacroix and Peyre (2005).

Usage

```
mgsim.cv(Ytrain,Xtrain,LambdaRange,hRange,NbIterMax=50)
```

Arguments

`Xtrain`	a (ntrain x p) data matrix of predictors. `Xtrain` must be a matrix. Each row corresponds to an observation and each column to a predictor variable.
`Ytrain`	a ntrain vector of responses. `Ytrain` must be a vector. `Ytrain` is a {1,...,c+1}-valued vector and contains the response variable for each observation. c+1 is the number of classes.
`LambdaRange`	the vector of positive real value from which the best ridge regularization parameter has to be chosen by cross-validation.
`hRange`	the vector of strictly positive real value from which the best bandwidth has to be chosen by cross-validation.
`NbIterMax`	a positive integer. `NbIterMax` is the maximal number of iterations in the Newton-Rapson parts.

Details

The cross-validation procedure described in Lambert-Lacroix and Peyre (2005) is used to determine the best ridge regularization parameter and bandwidth to be used for classification with GSIM for categorical data (for binary data see `gsim` and `gsim.cv`). At each cross-validation run, `Xtrain` is split into a pseudo training set (ntrain-1 samples) and a pseudo test set (1 sample) and the classification error rate is determined for each value of ridge regularization parameter and bandwidth. Finally, the function `mgsim.cv` returns the values of the ridge regularization parameter and bandwidth for which the mean classification error rate is minimal.

Value

A list with the following components:

`Lambda`	the optimal regularization parameter.
`h`	the optimal bandwidth parameter.

Author(s)

Sophie Lambert-Lacroix (http://membres-timc.imag.fr/Sophie.Lambert/) and Julie Peyre (http://www-lmc.imag.fr/lmc-sms/Julie.Peyre/).

References

S. Lambert-Lacroix, J. Peyre . (2006) Local likelyhood regression in generalized linear single-index models with applications to microarrays data. Computational Statistics and Data Analysis, vol 51, n 3, 2091-2113.

See Also

mgsim, gsim, gsim.cv.

Examples

```
## Not run:
## between 5~15 seconds
# load plsgenomics library
library(plsgenomics)

# load SRBCT data
data(SRBCT)
IndexLearn <- c(sample(which(SRBCT$Y==1),10),sample(which(SRBCT$Y==2),4),
                sample(which(SRBCT$Y==3),7),sample(which(SRBCT$Y==4),9))

### Determine optimum h and lambda
# /!\ take 30 secondes to run
#hl <- mgsim.cv(Ytrain=SRBCT$Y[IndexLearn],Xtrain=SRBCT$X[IndexLearn,],
#                            LambdaRange=c(0.1),hRange=c(7,20))

### perform prediction by MGSIM
#res <- mgsim(Ytrain=SRBCT$Y[IndexLearn],Xtrain=SRBCT$X[IndexLearn,],Lambda=hl$Lambda,
#             h=hl$h,Xtest=SRBCT$X[-IndexLearn,])
#res$Cvg
#sum(res$Ytest!=SRBCT$Y[-IndexLearn])


## End(Not run)
```

`mrpls` *Ridge Partial Least Square for categorical data*

Description

The function `mrpls` performs prediction using Fort et al. (2005) MRPLS algorithm.

Usage

```
mrpls(Ytrain,Xtrain,Lambda,ncomp,Xtest=NULL,NbIterMax=50)
```

Arguments

Xtrain	a (ntrain x p) data matrix of predictors. Xtrain must be a matrix. Each row corresponds to an observation and each column to a predictor variable.
Ytrain	a ntrain vector of responses. Ytrain must be a vector. Ytrain is a {0,...,c}-valued vector and contains the response variable for each observation. c+1 is the number of classes.
Xtest	a (ntest x p) matrix containing the predictors for the test data set. Xtest may also be a vector of length p (corresponding to only one test observation).If Xtest is not equal to NULL, then the prediction step is made for these new predictor variables.
Lambda	a positive real value. Lambda is the ridge regularization parameter.
ncomp	a positive integer. ncomp is the number of PLS components. If ncomp=0,then the Ridge regression is performed without reduction dimension.
NbIterMax	a positive integer. NbIterMax is the maximal number of iterations in the Newton-Rapson parts.

Details

The columns of the data matrices Xtrain and Xtest may not be standardized, since standardizing is performed by the function mrpls as a preliminary step before the algorithm is run.

The procedure described in Fort et al. (2005) is used to determine latent components to be used for classification and when Xtest is not equal to NULL, the procedure predicts the labels for these new predictor variables.

Value

A list with the following components:

Coefficients	the (p+1) x c matrix containing the coefficients weighting the block design matrix.
hatY	the ntrain vector containing the estimated {0,...,c}-valued labels for the observations from Xtrain.
hatYtest	the ntest vector containing the predicted {0,...,c}-valued labels for the observations from Xtest.
proba	the ntrain vector containing the estimated probabilities for the observations from Xtrain.
proba.test	the ntest vector containing the predicted probabilities for the observations from Xtest.
DeletedCol	the vector containing the column number of Xtrain when the variance of the corresponding predictor variable is null. Otherwise DeletedCol=NULL
hatYtest_k	If ncomp is greater than 1, hatYtest_k is a matrix of size ntest x ncomp in such a way that the kth column corresponds to the predicted label obtained with k PLS components.

Author(s)

Sophie Lambert-Lacroix (http://membres-timc.imag.fr/Sophie.Lambert/).

References

G. Fort, S. Lambert-Lacroix and Julie Peyre (2005). Reduction de dimension dans les modeles lineaires generalises : application a la classification supervisee de donnees issues des biopuces. Journal de la SFDS, tome 146, n1-2, 117-152.

See Also

mrpls.cv, rpls, rpls.cv.

Examples

```
# load plsgenomics library
library(plsgenomics)

# load SRBCT data
data(SRBCT)
IndexLearn <- c(sample(which(SRBCT$Y==1),10),sample(which(SRBCT$Y==2),4),
sample(which(SRBCT$Y==3),7),sample(which(SRBCT$Y==4),9))

# perform prediction by MRPLS
res <- mrpls(Ytrain=SRBCT$Y[IndexLearn]-1,Xtrain=SRBCT$X[IndexLearn,],Lambda=0.001,ncomp=2,
Xtest=SRBCT$X[-IndexLearn,])
sum(res$Ytest!=SRBCT$Y[-IndexLearn]-1)

# prediction for another sample
Xnew <- SRBCT$X[83,]
# Compute the linear predictor for each classes expect class 1
eta <- diag(t(cbind(c(1,Xnew),c(1,Xnew),c(1,Xnew))) %*% res$Coefficients)
Ypred <- which.max(c(0,eta))
Ypred+1
SRBCT$Y[83]
```

---

mrpls.cv — *Determination of the ridge regularization parameter and the number of PLS components to be used for classification with RPLS for categorical data*

Description

The function mrpls.cv determines the best ridge regularization parameter and the best number of PLS components to be used for classification for Fort et al. (2005) MRPLS algorithm.

Usage

```
mrpls.cv(Ytrain, Xtrain, LambdaRange, ncompMax, NbIterMax=50, ncores=1)
```

Arguments

Xtrain	a (ntrain x p) data matrix of predictors. Xtrain must be a matrix. Each row corresponds to an observation and each column to a predictor variable.
Ytrain	a ntrain vector of responses. Ytrain must be a vector. Ytrain is a {0,...,c}-valued vector and contains the response variable for each observation. c+1 is the number of classes.
LambdaRange	the vector of positive real value from which the best ridge regularization parameter has to be chosen by cross-validation.
ncompMax	a positive integer. The best number of components is chosen from 1,...,ncompMax. If ncompMax=0,then the Ridge regression is performed without reduction dimension.
NbIterMax	a positive integer. NbIterMax is the maximal number of iterations in the Newton-Rapson parts.
ncores	a positive integer. The number of cores to be used for parallel computing (if different from 1)

Details

A cross-validation procedure is used to determine the best ridge regularization parameter and number of PLS components to be used for classification with MRPLS for categorical data (for binary data see rpls and rpls.cv). At each cross-validation run, Xtrain is split into a pseudo training set (ntrain-1 samples) and a pseudo test set (1 sample) and the classification error rate is determined for each value of ridge regularization parameter and number of components. Finally, the function mrpls.cv returns the values of the ridge regularization parameter and bandwidth for which the mean classification error rate is minimal.

Value

A list with the following components:

Lambda	the optimal regularization parameter.
ncomp	the optimal number of PLS components.

Author(s)

Sophie Lambert-Lacroix (http://membres-timc.imag.fr/Sophie.Lambert/).

References

G. Fort, S. Lambert-Lacroix and Julie Peyre (2005). Reduction de dimension dans les modeles lineaires generalises : application a la classification supervisee de donnees issues des biopuces. Journal de la SFDS, tome 146, n1-2, 117-152.

See Also

mrpls, rpls, rpls.cv.

Examples

```
## Not run:
## between 5~15 seconds
# load plsgenomics library
library(plsgenomics)

# load SRBCT data
data(SRBCT)
IndexLearn <- c(sample(which(SRBCT$Y==1),10),sample(which(SRBCT$Y==2),4),
sample(which(SRBCT$Y==3),7),sample(which(SRBCT$Y==4),9))

# Determine optimum ncomp and Lambda
nl <- mrpls.cv(Ytrain=SRBCT$Y[IndexLearn]-1,Xtrain=SRBCT$X[IndexLearn,],
LambdaRange=c(0.1,1),ncompMax=3)

# perform prediction by MRPLS
res <- mrpls(Ytrain=SRBCT$Y[IndexLearn]-1,Xtrain=SRBCT$X[IndexLearn,],Lambda=nl$Lambda,
ncomp=nl$ncomp,Xtest=SRBCT$X[-IndexLearn,])
sum(res$Ytest!=SRBCT$Y[-IndexLearn]-1)

## End(Not run)
```

`multinom.spls` *Classification procedure for multi-label response based on a multinomial model, solved by a combination of the multinomial Ridge Iteratively Reweighted Least Squares (multinom-RIRLS) algorithm and the Adaptive Sparse PLS (SPLS) regression*

Description

The function `multinom.spls` performs compression and variable selection in the context of multi-label ('nclass' > 2) classification (with possible prediction) using Durif et al. (2017) algorithm based on Ridge IRLS and sparse PLS.

Usage

```
multinom.spls(Xtrain, Ytrain, lambda.ridge, lambda.l1, ncomp, Xtest = NULL,
  adapt = TRUE, maxIter = 100, svd.decompose = TRUE, center.X = TRUE,
  scale.X = FALSE, weighted.center = TRUE)
```

Arguments

`Xtrain`	a (ntrain x p) data matrix of predictor values. `Xtrain` must be a matrix. Each row corresponds to an observation and each column to a predictor variable.
`Ytrain`	a (ntrain) vector of (continuous) responses. `Ytrain` must be a vector or a one column matrix, and contains the response variable for each observation. `Ytrain` should take values in {0,...,nclass-1}, where nclass is the number of class.

lambda.ridge	a positive real value. lambda.ridge is the Ridge regularization parameter for the RIRLS algorithm (see details).
lambda.l1	a positive real value, in [0,1]. lambda.l1 is the sparse penalty parameter for the dimension reduction step by sparse PLS (see details).
ncomp	a positive integer. ncomp is the number of PLS components. If ncomp=0,then the Ridge regression is performed without any dimension reduction (no SPLS step).
Xtest	a (ntest x p) matrix containing the predictor values for the test data set. Xtest may also be a vector of length p (corresponding to only one test observation). Default value is NULL, meaning that no prediction is performed.
adapt	a boolean value, indicating whether the sparse PLS selection step sould be adaptive or not (see details).
maxIter	a positive integer. maxIter is the maximal number of iterations in the Newton-Raphson parts in the RIRLS algorithm (see details).
svd.decompose	a boolean parameter. svd.decompose indicates wether or not the predictor matrix Xtrain should be decomposed by SVD (singular values decomposition) for the RIRLS step (see details).
center.X	a boolean value indicating whether the data matrices Xtrain and Xtest (if provided) should be centered or not.
scale.X	a boolean value indicating whether the data matrices Xtrain and Xtest (if provided) should be scaled or not (scale.X=TRUE implies center.X=TRUE) in the spls step.
weighted.center	a boolean value indicating whether the centering should take into account the weighted l2 metric or not in the SPLS step.

Details

The columns of the data matrices Xtrain and Xtest may not be standardized, since standardizing can be performed by the function multinom.spls as a preliminary step.

The procedure described in Durif et al. (2017) is used to compute latent sparse components that are used in a multinomial regression model. In addition, when a matrix Xtest is supplied, the procedure predicts the response associated to these new values of the predictors.

Value

An object of class multinom.spls with the following attributes

Coefficients	a (p+1) x (nclass-1) matrix containing the linear coefficients associated to the predictors and intercept in the multinomial model explaining the response Y.
hatY	the (ntrain) vector containing the estimated response value on the train set Xtrain.
hatYtest	the (ntest) vector containing the predicted labels for the observations from Xtest (if provided).
DeletedCol	the vector containing the indexes of columns with null variance in Xtrain that were skipped in the procedure.

A	a list of size nclass-1 with predictors selected by the procedures for each set of coefficients in the multinomial model (i.e. indexes of the corresponding non null entries in each columns of Coefficients. Each elements of A is a subset of 1:p.
A.full	union of elements in A, corresponding to predictors selected in the full model.
Anames	Vector of selected predictor names, i.e. the names of the columns from Xtrain that are in A.full.
converged	a {0,1} value indicating whether the RIRLS algorithm did converge in less than maxIter iterations or not.
X.score	list of nclass-1 different (n x ncomp) matrices being the observations coordinates or scores in the new component basis produced for each class in the multinomial model by the SPLS step (sparse PLS), see Durif et al. (2017) for details.
X.weight	list of nclass-1 different (p x ncomp) matrices being the coefficients of predictors in each components produced for each class in the multinomial model by the sparse PLS, see Durif et al. (2017) for details.
X.score.full	a ((n x (nclass-1)) x ncomp) matrix being the observations coordinates or scores in the new component basis produced by the SPLS step (sparse PLS) in the linearized multinomial model, see Durif et al. (2017). Each column t.k of X.score is a SPLS component.
X.weight.full	a (p x ncomp) matrix being the coefficients of predictors in each components produced by sparse PLS in the linearized multinomial model, see Durif et al. (2017). Each column w.k of X.weight verifies t.k = Xtrain x w.k (as a matrix product).
lambda.ridge	the Ridge hyper-parameter used to fit the model.
lambda.l1	the sparse hyper-parameter used to fit the model.
ncomp	the number of components used to fit the model.
V	the (ntrain x ntrain) matrix used to weight the metric in the sparse PLS step. V is the inverse of the covariance matrix of the pseudo-response produced by the RIRLS step.
proba	the (ntrain) vector of estimated probabilities for the observations in code Xtrain, that are used to estimate the hatY labels.
proba.test	the (ntest) vector of predicted probabilities for the new observations in Xtest, that are used to predict the hatYtest labels.

Author(s)

Ghislain Durif (http://thoth.inrialpes.fr/people/gdurif/).

References

Durif G., Modolo L., Michaelsson J., Mold J. E., Lambert-Lacroix S., Picard F. (2017). High Dimensional Classification with combined Adaptive Sparse PLS and Logistic Regression, (in prep), available on (http://arxiv.org/abs/1502.05933).

See Also

spls, logit.spls, multinom.spls.cv

Examples

```
## Not run:
### load plsgenomics library
library(plsgenomics)

### generating data
n <- 100
p <- 100
nclass <- 3
sample1 <- sample.multinom(n, p, nb.class=nclass, kstar=20, lstar=2,
                           beta.min=0.25, beta.max=0.75,
                           mean.H=0.2, sigma.H=10, sigma.F=5)
X <- sample1$X
Y <- sample1$Y

### splitting between learning and testing set
index.train <- sort(sample(1:n, size=round(0.7*n)))
index.test <- (1:n)[-index.train]

Xtrain <- X[index.train,]
Ytrain <- Y[index.train,]
Xtest <- X[index.test,]
Ytest <- Y[index.test,]

### fitting the model, and predicting new observations
model1 <- multinom.spls(Xtrain=Xtrain, Ytrain=Ytrain, lambda.ridge=2,
                        lambda.l1=0.5, ncomp=2, Xtest=Xtest, adapt=TRUE,
                        maxIter=100, svd.decompose=TRUE)

str(model1)

### prediction error rate
sum(model1$hatYtest!=Ytest) / length(index.test)

## End(Not run)
```

`multinom.spls.cv` *Cross-validation procedure to calibrate the parameters (ncomp, lambda.l1, lambda.ridge) for the multinomial-SPLS method*

Description

The function `multinom.spls.cv` chooses the optimal values for the hyper-parameter of the `multinom.spls` procedure, by minimizing the averaged error of prediction over the hyper-parameter grid, using Durif et al. (2017) multinomial-SPLS algorithm.

Usage

```
multinom.spls.cv(X, Y, lambda.ridge.range, lambda.l1.range, ncomp.range,
  adapt = TRUE, maxIter = 100, svd.decompose = TRUE,
  return.grid = FALSE, ncores = 1, nfolds = 10, nrun = 1,
  center.X = TRUE, scale.X = FALSE, weighted.center = TRUE, seed = NULL,
  verbose = TRUE)
```

Arguments

X	a (n x p) data matrix of predictors. X must be a matrix. Each row corresponds to an observation and each column to a predictor variable.
Y	a (n) vector of (continuous) responses. Y must be a vector or a one column matrix. It contains the response variable for each observation. Y should take values in {0,...,nclass-1}, where nclass is the number of class.
lambda.ridge.range	a vector of positive real values. lambda.ridge is the Ridge regularization parameter for the RIRLS algorithm (see details), the optimal value will be chosen among lambda.ridge.range.
lambda.l1.range	a vecor of positive real values, in [0,1]. lambda.l1 is the sparse penalty parameter for the dimension reduction step by sparse PLS (see details), the optimal value will be chosen among lambda.l1.range.
ncomp.range	a vector of positive integers. ncomp is the number of PLS components. The optimal value will be chosen among ncomp.range.
adapt	a boolean value, indicating whether the sparse PLS selection step sould be adaptive or not (see details).
maxIter	a positive integer, the maximal number of iterations in the RIRLS algorithm (see details).
svd.decompose	a boolean parameter. svd.decompose indicates wether or not the predictor matrix Xtrain should be decomposed by SVD (singular values decomposition) for the RIRLS step (see details).
return.grid	a boolean values indicating whether the grid of hyper-parameters values with corresponding mean prediction error rate over the folds should be returned or not.
ncores	a positve integer, indicating the number of cores that the cross-validation is allowed to use for parallel computation (see details).
nfolds	a positive integer indicating the number of folds in the K-folds cross-validation procedure, nfolds=n corresponds to the leave-one-out cross-validation, default is 10.
nrun	a positive integer indicating how many times the K-folds cross- validation procedure should be repeated, default is 1.
center.X	a boolean value indicating whether the data matrices Xtrain and Xtest (if provided) should be centered or not.

scale.X	a boolean value indicating whether the data matrices Xtrain and Xtest (if provided) should be scaled or not (scale.X=TRUE implies center.X=TRUE) in the spls step.
weighted.center	a boolean value indicating whether the centering should take into account the weighted l2 metric or not in the SPLS step.
seed	a positive integer value (default is NULL). If non NULL, the seed for pseudo-random number generation is set accordingly.
verbose	a boolean parameter indicating the verbosity.

Details

The columns of the data matrices X may not be standardized, since standardizing is performed by the function multinom.spls.cv as a preliminary step.

The procedure is described in Durif et al. (2017). The K-fold cross-validation can be summarize as follow: the train set is partitioned into K folds, for each value of hyper-parameters the model is fit K times, using each fold to compute the prediction error rate, and fitting the model on the remaining observations. The cross-validation procedure returns the optimal hyper-parameters values, meaning the one that minimize the averaged error of prediction averaged over all the folds.

This procedures uses mclapply from the parallel package, available on GNU/Linux and MacOS. Users of Microsoft Windows can refer to the README file in the source to be able to use a mclapply type function.

Value

An object of class multinom.spls with the following attributes

lambda.ridge.opt	the optimal value in lambda.ridge.range.
lambda.l1.opt	the optimal value in lambda.l1.range.
ncomp.opt	the optimal value in ncomp.range.
conv.per	the overall percentage of models that converge during the cross-validation procedure.
cv.grid	the grid of hyper-parameters and corresponding prediction error rate averaged over the folds. cv.grid is NULL if return.grid is set to FALSE.

Author(s)

Ghislain Durif (http://thoth.inrialpes.fr/people/gdurif/).

References

Durif G., Modolo L., Michaelsson J., Mold J. E., Lambert-Lacroix S., Picard F. (2017). High Dimensional Classification with combined Adaptive Sparse PLS and Logistic Regression, (in prep), available on (http://arxiv.org/abs/1502.05933).

See Also

multinom.spls, multinom.spls.stab

Examples

```
## Not run:
### load plsgenomics library
library(plsgenomics)

### generating data
n <- 100
p <- 100
nclass <- 3
sample1 <- sample.multinom(n=n, p=p, nb.class=nclass, kstar=10, lstar=2,
                           beta.min=0.25, beta.max=0.75, mean.H=0.2,
                           sigma.H=10, sigma.F=5)

X <- sample1$X
Y <- sample1$Y

### hyper-parameters values to test
lambda.l1.range <- seq(0.05,0.95,by=0.1) # between 0 and 1
ncomp.range <- 1:10
# log-linear range between 0.01 a,d 1000 for lambda.ridge.range
logspace <- function( d1, d2, n) exp(log(10)*seq(d1, d2, length.out=n))
lambda.ridge.range <- signif(logspace(d1 <- -2, d2 <- 3, n=21), digits=3)

### tuning the hyper-parameters
cv1 <- multinom.spls.cv(X=X, Y=Y, lambda.ridge.range=lambda.ridge.range,
                        lambda.l1.range=lambda.l1.range,
                        ncomp.range=ncomp.range,
                        adapt=TRUE, maxIter=100, svd.decompose=TRUE,
                        return.grid=TRUE, ncores=1, nfolds=10)

str(cv1)

## End(Not run)
```

`multinom.spls.stab` *Stability selection procedure to estimate probabilities of selection of covariates for the multinomial-SPLS method*

Description

The function `multinom.spls.stab` train a multinomial-spls model for each candidate values (`ncomp`, `lambda.l1`, `lambda.ridge`) of hyper-parameters on multiple sub-samplings in the data. The stability selection procedure selects the covariates that are selected by most of the models among

the grid of hyper-parameters, following the procedure described in Durif et al. (2017). Candidates values for ncomp, lambda.l1 and lambda.l2 are respectively given by the input arguments ncomp.range, lambda.l1.range and lambda.l2.range.

Usage

```
multinom.spls.stab(X, Y, lambda.ridge.range, lambda.l1.range, ncomp.range,
  adapt = TRUE, maxIter = 100, svd.decompose = TRUE, ncores = 1,
  nresamp = 100, center.X = TRUE, scale.X = FALSE,
  weighted.center = TRUE, seed = NULL, verbose = TRUE)
```

Arguments

X	a (n x p) data matrix of predictors. X must be a matrix. Each row corresponds to an observation and each column to a predictor variable.
Y	a (n) vector of (continuous) responses. Y must be a vector or a one column matrix. It contains the response variable for each observation. Y should take values in {0,...,nclass-1}, where nclass is the number of class.
lambda.ridge.range	a vector of positive real values. lambda.ridge is the Ridge regularization parameter for the RIRLS algorithm (see details), the optimal value will be chosen among lambda.ridge.range.
lambda.l1.range	a vecor of positive real values, in [0,1]. lambda.l1 is the sparse penalty parameter for the dimension reduction step by sparse PLS (see details), the optimal value will be chosen among lambda.l1.range.
ncomp.range	a vector of positive integers. ncomp is the number of PLS components. The optimal value will be chosen among ncomp.range.
adapt	a boolean value, indicating whether the sparse PLS selection step sould be adaptive or not (see details).
maxIter	a positive integer, the maximal number of iterations in the RIRLS algorithm (see details).
svd.decompose	a boolean parameter. svd.decompose indicates wether or not the predictor matrix Xtrain should be decomposed by SVD (singular values decomposition) for the RIRLS step (see details).
ncores	a positve integer, indicating the number of cores that the cross-validation is allowed to use for parallel computation (see details).
nresamp	number of resamplings of the data to estimate the probility of selection for each covariate, default is 100.
center.X	a boolean value indicating whether the data matrices Xtrain and Xtest (if provided) should be centered or not.
scale.X	a boolean value indicating whether the data matrices Xtrain and Xtest (if provided) should be scaled or not (scale.X=TRUE implies center.X=TRUE) in the spls step.

`weighted.center`
: a boolean value indicating whether the centering should take into account the weighted l2 metric or not in the SPLS step.

`seed`
: a positive integer value (default is NULL). If non NULL, the seed for pseudo-random number generation is set accordingly.

`verbose`
: a boolean parameter indicating the verbosity.

Details

The columns of the data matrices `X` may not be standardized, since standardizing is performed by the function `multinom.spls.stab` as a preliminary step.

The procedure is described in Durif et al. (2017). The stability selection procedure can be summarize as follow (c.f. Meinshausen and Buhlmann, 2010).

(i) For each candidate values `(ncomp, lambda.l1, lambda.ridge)` of hyper-parameters, a multinomial-spls is trained on `nresamp` resamplings of the data. Then, for each triplet `(ncomp, lambda.l1, lambda.ridge)`, the probability that a covariate (i.e. a column in `X`) is selected is computed among the resamplings.

The estimated probabilities can be visualized as a heatmap with the function `stability.selection.heatmap`.

(ii) Eventually, the set of "stable selected" variables corresponds to the set of covariates that were selected by most of the training among the grid of hyper-parameters candidate values.

This function achieves the first step (i) of the stability selection procedure. The second step (ii) is achieved by the function `stability.selection`

This procedures uses `mclapply` from the `parallel` package, available on GNU/Linux and MacOS. Users of Microsoft Windows can refer to the README file in the source to be able to use a mclapply type function.

Value

An object with the following attributes

`q.Lambda`
: A table with values of q.Lambda (c.f. Durif et al. (2017) for the notation), being the averaged number of covariates selected among the entire grid of hyper-parameters candidates values, for increasing size of hyper-parameter grid.

`probs.lambda`
: A table with estimated probability of selection for each covariates depending on the candidates values for hyper-parameters.

`p`
: An integer values indicating the number of covariates in the model.

Author(s)

Ghislain Durif (http://thoth.inrialpes.fr/people/gdurif/).

References

Durif G., Modolo L., Michaelsson J., Mold J. E., Lambert-Lacroix S., Picard F. (2017). High Dimensional Classification with combined Adaptive Sparse PLS and Logistic Regression, (in prep), available on (http://arxiv.org/abs/1502.05933).

Meinshausen, N., Buhlmann P. (2010). Stability Selection. Journal of the Royal Statistical Society: Series B (Statistical Methodology) 72, no. 4, 417-473.

See Also

multinom.spls, stability.selection, stability.selection.heatmap

Examples

```
## Not run:
### load plsgenomics library
library(plsgenomics)

### generating data
n <- 100
p <- 100
nclass <- 3
sample1 <- sample.multinom(n, p, nb.class=nclass, kstar=20, lstar=2,
                           beta.min=0.25, beta.max=0.75,
                           mean.H=0.2, sigma.H=10, sigma.F=5)

X <- sample1$X
Y <- sample1$Y

### pertinent covariates id
sample1$sel

### hyper-parameters values to test
lambda.l1.range <- seq(0.05,0.95,by=0.1) # between 0 and 1
ncomp.range <- 1:10
# log-linear range between 0.01 a,d 1000 for lambda.ridge.range
logspace <- function( d1, d2, n) exp(log(10)*seq(d1, d2, length.out=n))
lambda.ridge.range <- signif(logspace(d1 <- -2, d2 <- 3, n=21), digits=3)

### tuning the hyper-parameters
stab1 <- multinom.spls.stab(X=X, Y=Y, lambda.ridge.range=lambda.ridge.range,
                            lambda.l1.range=lambda.l1.range,
                            ncomp.range=ncomp.range,
                            adapt=TRUE, maxIter=100, svd.decompose=TRUE,
                            ncores=1, nresamp=100)

str(stab1)

### heatmap of estimated probabilities
stability.selection.heatmap(stab1)

### selected covariates
stability.selection(stab1, piThreshold=0.6, rhoError=10)

## End(Not run)
```

pls.lda *Classification with PLS Dimension Reduction and Linear Discriminant Analysis*

Description

The function pls.lda performs binary or multicategorical classification using the method described in Boulesteix (2004) which consists in PLS dimension reduction and linear discriminant analysis applied on the PLS components.

Usage

```
pls.lda(Xtrain, Ytrain, Xtest=NULL, ncomp, nruncv=0, alpha=2/3, priors=NULL)
```

Arguments

Xtrain	a (ntrain x p) data matrix containing the predictors for the training data set. Xtrain may be a matrix or a data frame. Each row is an observation and each column is a predictor variable.
Ytrain	a vector of length ntrain giving the classes of the ntrain observations. The classes must be coded as 1,...,K (K>=2).
Xtest	a (ntest x p) data matrix containing the predictors for the test data set. Xtest may also be a vector of length p (corresponding to only one test observation). If Xtest=NULL, the training data set is considered as test data set as well.
ncomp	if nruncv=0, ncomp is the number of latent components to be used for PLS dimension reduction. If nruncv>0, the cross-validation procedure described in Boulesteix (2004) is used to choose the best number of components from the vector of integers ncomp or from 1,...,ncomp if ncomp is of length 1.
nruncv	the number of cross-validation iterations to be performed for the choice of the number of latent components. If nruncv=0, cross-validation is not performed and ncomp latent components are used.
alpha	the proportion of observations to be included in the training set at each cross-validation iteration.
priors	The class priors to be used for linear discriminant analysis. If unspecified, the class proportions in the training set are used.

Details

The function pls.lda proceeds as follows to predict the class of the observations from the test data set. First, the SIMPLS algorithm is run on Xtrain and Ytrain to determine the new PLS components based on the training observations only. The new PLS components are then computed for the test data set. Classification is performed by applying classical linear discriminant analysis (LDA) to the new components. Of course, the LDA classifier is built using the training observations only.

Value

A list with the following components:

predclass	the vector containing the predicted classes of the ntest observations from Xtest.
ncomp	the number of latent components used for classification.
pls.out	an object containing the results from the call of the pls.regression function (from the plsgenomics package).
lda.out	an object containing the results from the call of the lda function (from the MASS package).
pred.lda.out	an object containing the results from the call of the predict.lda function (from the MASS package).

Author(s)

Anne-Laure Boulesteix (http://www.ibe.med.uni-muenchen.de/organisation/mitarbeiter/020_professuren/boulesteix/eng.html)

References

A. L. Boulesteix (2004). PLS dimension reduction for classification with microarray data, Statistical Applications in Genetics and Molecular Biology **3**, Issue 1, Article 33.

A. L. Boulesteix, K. Strimmer (2007). Partial least squares: a versatile tool for the analysis of high-dimensional genomic data. Briefings in Bioinformatics 7:32-44.

S. de Jong (1993). SIMPLS: an alternative approach to partial least squares regression, Chemometrics Intell. Lab. Syst. **18**, 251–263.

See Also

pls.regression, variable.selection, pls.lda.cv.

Examples

```
# load plsgenomics library
library(plsgenomics)

# load leukemia data
data(leukemia)

# Classify observations 1,2,3 (test set) using observations 4 to 38 (training set),
# with 2 PLS components
pls.lda(Xtrain=leukemia$X[-(1:3),],Ytrain=leukemia$Y[-(1:3)],Xtest=leukemia$X[1:3,],
      ncomp=2,nruncv=0)

# Classify observations 1,2,3 (test set) using observations 4 to 38 (training set),
# with the best number of components as determined by cross-validation
pls.lda(Xtrain=leukemia$X[-(1:3),],Ytrain=leukemia$Y[-(1:3)],Xtest=leukemia$X[1:3,],
ncomp=1:4,nruncv=20)
```

`pls.lda.cv` *Determination of the number of latent components to be used for classification with PLS and LDA*

Description

The function `pls.lda.cv` determines the best number of latent components to be used for classification with PLS dimension reduction and linear discriminant analysis as described in Boulesteix (2004).

Usage

```
pls.lda.cv(Xtrain, Ytrain,  ncomp, nruncv=20, alpha=2/3, priors=NULL)
```

Arguments

`Xtrain`	a (ntrain x p) data matrix containing the predictors for the training data set. `Xtrain` may be a matrix or a data frame. Each row is an observation and each column is a predictor variable.
`Ytrain`	a vector of length ntrain giving the classes of the ntrain observations. The classes must be coded as 1,...,K (K>=2).
`ncomp`	the vector of integers from which the best number of latent components has to be chosen by cross-validation. If `ncomp` is of length 1, the best number of components is chosen from 1,...,`ncomp`.
`nruncv`	the number of cross-validation iterations to be performed for the choice of the number of latent components.
`alpha`	the proportion of observations to be included in the training set at each cross-validation iteration.
`priors`	The class priors to be used for linear discriminant analysis. If unspecified, the class proportions in the training set are used.

Details

The cross-validation procedure described in Boulesteix (2004) is used to determine the best number of latent components to be used for classification. At each cross-validation run, `Xtrain` is split into a pseudo training set and a pseudo test set and the classification error rate is determined for each number of latent components. Finally, the function `pls.lda.cv` returns the number of latent components for which the mean classification rate over the `nrun` partitions is minimal.

Value

The number of latent components to be used for classification.

Author(s)

Anne-Laure Boulesteix (http://www.ibe.med.uni-muenchen.de/organisation/mitarbeiter/020_professuren/boulesteix/eng.html)

References

A. L. Boulesteix (2004). PLS dimension reduction for classification with microarray data, Statistical Applications in Genetics and Molecular Biology **3**, Issue 1, Article 33.

A. L. Boulesteix, K. Strimmer (2007). Partial least squares: a versatile tool for the analysis of high-dimensional genomic data. Briefings in Bioinformatics 7:32-44.

S. de Jong (1993). SIMPLS: an alternative approach to partial least squares regression, Chemometrics Intell. Lab. Syst. **18**, 251–263.

See Also

pls.lda, pls.regression.cv.

Examples

```
## Not run:
## between 5~15 seconds
# load plsgenomics library
library(plsgenomics)

# load leukemia data
data(leukemia)

# Determine the best number of components to be used for classification using the
# cross-validation procedure
# choose the best number from 2,3,4
pls.lda.cv(Xtrain=leukemia$X,Ytrain=leukemia$Y,ncomp=2:4,nruncv=20)
# choose the best number from 1,2,3
pls.lda.cv(Xtrain=leukemia$X,Ytrain=leukemia$Y,ncomp=3,nruncv=20)


## End(Not run)
```

---

pls.regression *Multivariate Partial Least Squares Regression*

Description

The function pls.regression performs pls multivariate regression (with several response variables and several predictor variables) using de Jong's SIMPLS algorithm. This function is an adaptation of R. Wehrens' code from the package pls.pcr.

Usage

```
pls.regression(Xtrain, Ytrain, Xtest=NULL, ncomp=NULL,  unit.weights=TRUE)
```

Arguments

Xtrain	a (ntrain x p) data matrix of predictors. Xtrain may be a matrix or a data frame. Each row corresponds to an observation and each column to a predictor variable.
Ytrain	a (ntrain x m) data matrix of responses. Ytrain may be a vector (if m=1), a matrix or a data frame. If Ytrain is a matrix or a data frame, each row corresponds to an observation and each column to a response variable. If Ytrain is a vector, it contains the unique response variable for each observation.
Xtest	a (ntest x p) matrix containing the predictors for the test data set. Xtest may also be a vector of length p (corresponding to only one test observation).
ncomp	the number of latent components to be used for regression. If ncomp is a vector of integers, the regression model is built successively with each number of components. If ncomp=NULL, the maximal number of components min(ntrain,p) is chosen.
unit.weights	if TRUE then the latent components will be constructed from weight vectors that are standardized to length 1, otherwise the weight vectors do not have length 1 but the latent components have norm 1.

Details

The columns of the data matrices Xtrain and Ytrain must not be centered to have mean zero, since centering is performed by the function pls.regression as a preliminary step before the SIMPLS algorithm is run.

In the original definition of SIMPLS by de Jong (1993), the weight vectors have length 1. If the weight vectors are standardized to have length 1, they satisfy a simple optimality criterion (de Jong, 1993). However, it is also usual (and computationally efficient) to standardize the latent components to have length 1.

In contrast to the original version found in the package pls.pcr, the prediction for the observations from Xtest is performed after centering the columns of Xtest by substracting the columns means calculated from Xtrain.

Value

A list with the following components:

B	the (p x m x length(ncomp)) matrix containing the regression coefficients. Each row corresponds to a predictor variable and each column to a response variable. The third dimension of the matrix B corresponds to the number of PLS components used to compute the regression coefficients. If ncomp has length 1, B is just a (p x m) matrix.
Ypred	the (ntest x m x length(ncomp)) containing the predicted values of the response variables for the observations from Xtest. The third dimension of the matrix Ypred corresponds to the number of PLS components used to compute the regression coefficients.
P	the (p x max(ncomp)) matrix containing the X-loadings.
Q	the (m x max(ncomp)) matrix containing the Y-loadings.
T	the (ntrain x max(ncomp)) matrix containing the X-scores (latent components)

R	the (p x max(ncomp)) matrix containing the weights used to construct the latent components.
meanX	the p-vector containing the means of the columns of Xtrain.

Author(s)

Anne-Laure Boulesteix (http://www.ibe.med.uni-muenchen.de/organisation/mitarbeiter/020_professuren/boulesteix/eng.html) and Korbinian Strimmer (http://strimmerlab.org/).

Adapted in part from pls.pcr code by R. Wehrens (in a former version of the 'pls' package https://CRAN.R-project.org/package=pls).

References

S. de Jong (1993). SIMPLS: an alternative approach to partial least squares regression, Chemometrics Intell. Lab. Syst. **18**, 251–263.

C. J. F. ter Braak and S. de Jong (1993). The objective function of partial least squares regression, Journal of Chemometrics **12**, 41–54.

See Also

pls.lda, TFA.estimate, pls.regression.cv.

Examples

```
# load plsgenomics library
library(plsgenomics)

# load the Ecoli data
data(Ecoli)

# perform pls regression
# with unit latent components
pls.regression(Xtrain=Ecoli$CONNECdata,Ytrain=Ecoli$GEdata,Xtest=Ecoli$CONNECdata,
ncomp=1:3,unit.weights=FALSE)

# with unit weight vectors
pls.regression(Xtrain=Ecoli$CONNECdata,Ytrain=Ecoli$GEdata,Xtest=Ecoli$CONNECdata,
ncomp=1:3,unit.weights=TRUE)
```

---

pls.regression.cv *Determination of the number of latent components to be used in PLS regression*

Description

The function pls.regression.cv determines the best number of latent components to be used for PLS regression using the cross-validation approach described in Boulesteix and Strimmer (2005).

Usage

```
pls.regression.cv(Xtrain, Ytrain, ncomp, nruncv=20, alpha=2/3)
```

Arguments

Xtrain	a (ntrain x p) data matrix containing the predictors for the training data set. Xtrain may be a matrix or a data frame. Each row is an observation and each column is a predictor variable.
Ytrain	a (ntrain x m) data matrix of responses. Ytrain may be a vector (if m=1), a matrix or a data frame. If Ytrain is a matrix or a data frame, each row is an observation and each column is a response variable. If Ytrain is a vector, it contains the unique response variable for each observation.
ncomp	the vector of integers from which the best number of latent components has to be chosen by cross-validation. If ncomp is of length 1, the best number of components is chosen from 1,...,ncomp.
nruncv	the number of cross-validation iterations to be performed for the choice of the number of latent components.
alpha	the proportion of observations to be included in the training set at each cross-validation iteration.

Details

The cross-validation procedure described in Boulesteix and Strimmer (2005) is used to determine the best number of latent components to be used for classification. At each cross-validation run, Xtrain is split into a pseudo training set and a pseudo test set and the squared error is determined for each number of latent components. Finally, the function pls.regression.cv returns the number of latent components for which the mean squared error over the nrun partitions is minimal.

Value

The number of latent components to be used in PLS regression, as determined by cross-validation.

Author(s)

Anne-Laure Boulesteix (http://www.ibe.med.uni-muenchen.de/organisation/mitarbeiter/020_professuren/boulesteix/eng.html) and Korbinian Strimmer (http://strimmerlab.org/).

References

A. L. Boulesteix and K. Strimmer (2005). Predicting Transcription Factor Activities from Combined Analysis of Microarray and ChIP Data: A Partial Least Squares Approach.

A. L. Boulesteix, K. Strimmer (2007). Partial least squares: a versatile tool for the analysis of high-dimensional genomic data. Briefings in Bioinformatics 7:32-44.

S. de Jong (1993). SIMPLS: an alternative approach to partial least squares regression, Chemometrics Intell. Lab. Syst. **18**, 251–263.

See Also

pls.regression, TFA.estimate, pls.lda.cv.

Examples

```
## Not run:
## between 5~15 seconds
# load plsgenomics library
library(plsgenomics)

# load Ecoli data
data(Ecoli)

# determine the best number of components for PLS regression using the cross-validation approach
# choose the best number from 1,2,3,4
pls.regression.cv(Xtrain=Ecoli$CONNECdata,Ytrain=Ecoli$GEdata,ncomp=4,nruncv=20)
# choose the best number from 2,3
pls.regression.cv(Xtrain=Ecoli$CONNECdata,Ytrain=Ecoli$GEdata,ncomp=c(2,3),nruncv=20)


## End(Not run)
```

---

plsgenomics-deprecated

Deprecated function(s) in the 'plsgenomics' package

Description

These functions are provided for compatibility with older version of the 'plsgenomics' package. They may eventually be completely removed.

Usage

```
m.rirls.spls(...)
```

Arguments

...	Parameters to be passed to the modern version of the function

Details

rirls.spls	is replaced by logit.spls
rirls.spls.tune	is replaced by logit.spls.cv
rirls.spls.stab	is replaced by logit.spls.stab
m.rirls.spls	is replaced by multinom.spls
m.rirls.spls.tune	is replaced by multinom.spls.cv
m.rirls.spls.stab	is replaced by multinom.spls.stab
spls.adapt	is replaced by spls
spls.adapt.tune	is replaced by spls.cv

`preprocess` *preprocess for microarray data*

Description

The function `preprocess` performs a preprocessing of microarray data.

Usage

```
preprocess(Xtrain, Xtest=NULL,Threshold=c(100,16000),Filtering=c(5,500),
log10.scale=TRUE,row.stand=TRUE)
```

Arguments

`Xtrain`	a (ntrain x p) data matrix of predictors. `Xtrain` must be a matrix. Each row corresponds to an observation and each column to a predictor variable.
`Xtest`	a (ntest x p) matrix containing the predictors for the test data set. `Xtest` may also be a vector of length p (corresponding to only one test observation).
`Threshold`	a vector of length 2 containing the values (threshmin,threshmax) for thresholding data in preprocess. Data is thresholded to value threshmin and ceiled to value threshmax. If `Threshold` is NULL then no thresholding is done. By default, if the value given for `Threshold` is not valid, no thresholding is done.
`Filtering`	a vector of length 2 containing the values (FiltMin,FiltMax) for filtering genes in preprocess. Genes with max/min$<= FiltMin$ and (max-min)$<= FiltMax$ are excluded. If `Filtering` is NULL then no thresholding is done. By default, if the value given for `Filtering` is not valid, no filtering is done.
`log10.scale`	a logical value equal to TRUE if a log10-transformation has to be done.
`row.stand`	a logical value equal to TRUE if a standardisation in row has to be done.

Details

The pre-processing steps recommended by Dudoit et al. (2002) are performed. The default values are those adapted for Colon data.

Value

A list with the following components:

`pXtrain`	the (ntrain x p') matrix containing the preprocessed train data.
`pXtest`	the (ntest x p') matrix containing the preprocessed test data.

Author(s)

Sophie Lambert-Lacroix (http://membres-timc.imag.fr/Sophie.Lambert/) and Julie Peyre (http://www-lmc.imag.fr/lmc-sms/Julie.Peyre/).

References

Dudoit, S. and Fridlyand, J. and Speed, T. (2002). Comparison of discrimination methods for the classification of tumors using gene expression data, Journal of the American Statistical Association, 97, 77–87.

Examples

```
# load plsgenomics library
library(plsgenomics)

# load Colon data
data(Colon)
IndexLearn <- c(sample(which(Colon$Y==2),27),sample(which(Colon$Y==1),14))

Xtrain <- Colon$X[IndexLearn,]
Ytrain <- Colon$Y[IndexLearn]
Xtest <- Colon$X[-IndexLearn,]

# preprocess data
resP <- preprocess(Xtrain= Xtrain, Xtest=Xtest,Threshold = c(100,16000),Filtering=c(5,500),
log10.scale=TRUE,row.stand=TRUE)

# how many genes after preprocess ?
dim(resP$pXtrain)[2]
```

---

rpls *Ridge Partial Least Square for binary data*

Description

The function `mrpls` performs prediction using Fort and Lambert-Lacroix (2005) RPLS algorithm.

Usage

```
rpls(Ytrain,Xtrain,Lambda,ncomp,Xtest=NULL,NbIterMax=50)
```

Arguments

`Xtrain`	a (ntrain x p) data matrix of predictors. `Xtrain` must be a matrix. Each row corresponds to an observation and each column to a predictor variable.
`Ytrain`	a ntrain vector of responses. `Ytrain` must be a vector. `Ytrain` is a {0,1}-valued vector and contains the response variable for each observation.
`Xtest`	a (ntest x p) matrix containing the predictors for the test data set. `Xtest` may also be a vector of length p (corresponding to only one test observation).If `Xtest` is not equal to NULL, then the prediction step is made for these new predictor variables.
`Lambda`	a positive real value. `Lambda` is the ridge regularization parameter.

ncomp	a positive integer. ncomp is the number of PLS components. If ncomp=0,then the Ridge regression is performed without reduction dimension.
NbIterMax	a positive integer. NbIterMax is the maximal number of iterations in the Newton-Rapson parts.

Details

The columns of the data matrices Xtrain and Xtest may not be standardized, since standardizing is performed by the function rpls as a preliminary step before the algorithm is run.

The procedure described in Fort and Lambert-Lacroix (2005) is used to determine latent components to be used for classification and when Xtest is not equal to NULL, the procedure predicts the labels for these new predictor variables.

Value

A list with the following components:

Coefficients	the (p+1) vector containing the coefficients weighting the design matrix.
hatY	the ntrain vector containing the estimated {0,1}-valued labels for the observations from Xtrain.
hatYtest	the ntest vector containing the predicted {0,1}-valued labels for the observations from Xtest.
proba	the ntrain vector containing the estimated probabilities for the observations from Xtrain.
proba.test	the ntest vector containing the predicted probabilities for the observations from Xtest.
DeletedCol	the vector containing the column number of Xtrain when the variance of the corresponding predictor variable is null. Otherwise DeletedCol=NULL
hatYtest_k	If ncomp is greater than 1, hatYtest_k is a {0,1}-valued matrix of size ntest x ncomp in such a way that the kth column corresponds to the predicted label obtained with k PLS components.

Author(s)

Sophie Lambert-Lacroix (http://membres-timc.imag.fr/Sophie.Lambert/).

References

G. Fort and S. Lambert-Lacroix (2005). Classification using Partial Least Squares with Penalized Logistic Regression, Bioinformatics, vol 21, n 8, 1104-1111.

See Also

rpls.cv, mrpls, mrpls.cv.

Examples

```
# load plsgenomics library
library(plsgenomics)

# load Colon data
data(Colon)
IndexLearn <- c(sample(which(Colon$Y==2),12),sample(which(Colon$Y==1),8))

# preprocess data
res <- preprocess(Xtrain= Colon$X[IndexLearn,], Xtest=Colon$X[-IndexLearn,],
                  Threshold = c(100,16000),Filtering=c(5,500),
                  log10.scale=TRUE,row.stand=TRUE)
# the results are given in res$pXtrain and res$pXtest

# perform prediction by RPLS
resrpls <- rpls(Ytrain=Colon$Y[IndexLearn]-1,Xtrain=res$pXtrain,Lambda=0.6,ncomp=1,Xtest=res$pXtest)
resrpls$hatY
sum(resrpls$Ytest!=Colon$Y[-IndexLearn])

# prediction for another sample
Xnew <- res$pXtest[1,]
# Compute the linear predictor for each classes expect class 0
eta <- c(1,Xnew) %*% resrpls$Coefficients
Ypred <- which.max(c(0,eta))
Ypred+1
```

`rpls.cv` *Determination of the ridge regularization parameter and the number of PLS components to be used for classification with RPLS for binary data*

Description

The function `rpls.cv` determines the best ridge regularization parameter and the best number of PLS components to be used for classification for Fort and Lambert-Lacroix (2005) RPLS algorithm.

Usage

```
rpls.cv(Ytrain, Xtrain, LambdaRange, ncompMax, NbIterMax=50, ncores=1)
```

Arguments

`Xtrain`	a (ntrain x p) data matrix of predictors. `Xtrain` must be a matrix. Each row corresponds to an observation and each column to a predictor variable.
`Ytrain`	a ntrain vector of responses. `Ytrain` must be a vector. `Ytrain` is a {0,1}-valued vector and contains the response variable for each observation.

LambdaRange	the vector of positive real value from which the best ridge regularization parameter has to be chosen by cross-validation.
ncompMax	a positive integer. the best number of components is chosen from 1,...,ncompMax. If ncompMax=0,then the Ridge regression is performed without reduction dimension.
NbIterMax	a positive integer. NbIterMax is the maximal number of iterations in the Newton-Rapson parts.
ncores	a positive integer. The number of cores to be used for parallel computing (if different from 1)

Details

A cross-validation procedure is used to determine the best ridge regularization parameter and number of PLS components to be used for classification with RPLS for binary data (for categorical data see mrpls and mrpls.cv). At each cross-validation run, Xtrain is split into a pseudo training set (ntrain-1 samples) and a pseudo test set (1 sample) and the classification error rate is determined for each value of ridge regularization parameter and number of components. Finally, the function mrpls.cv returns the values of the ridge regularization parameter and bandwidth for which the mean classification error rate is minimal.

Value

A list with the following components:

Lambda	the optimal regularization parameter.
ncomp	the optimal number of PLS components.

Author(s)

Sophie Lambert-Lacroix (http://membres-timc.imag.fr/Sophie.Lambert/).

References

G. Fort and S. Lambert-Lacroix (2005). Classification using Partial Least Squares with Penalized Logistic Regression, Bioinformatics, vol 21, n 8, 1104-1111.

See Also

rpls, mrpls, mrpls.cv.

Examples

```
## Not run:
## between 5~15 seconds
# load plsgenomics library
library(plsgenomics)

# load Colon data
data(Colon)
IndexLearn <- c(sample(which(Colon$Y==2),12),sample(which(Colon$Y==1),8))
```

```
# preprocess data
res <- preprocess(Xtrain= Colon$X[IndexLearn,], Xtest=Colon$X[-IndexLearn,],
                    Threshold = c(100,16000),Filtering=c(5,500),
                    log10.scale=TRUE,row.stand=TRUE)
# the results are given in res$pXtrain and res$pXtest

# Determine optimum ncomp and lambda
nl <- rpls.cv(Ytrain=Colon$Y[IndexLearn]-1,Xtrain=res$pXtrain,LambdaRange=c(0.1,1),ncompMax=3)

# perform prediction by RPLS
resrpls <- rpls(Ytrain=Colon$Y[IndexLearn]-1,Xtrain=res$pXtrain,Lambda=nl$Lambda,
ncomp=nl$ncomp,Xtest=res$pXtest)
sum(resrpls$Ytest!=Colon$Y[-IndexLearn]-1)


## End(Not run)
```

`sample.bin` *Generates covariate matrix X with correlated block of covariates and a binary random reponse depening on X through a logistic model*

Description

The function `sample.bin` generates a random sample of n observations, composed of p predictors, collected in the n x p matrix X, and a binary response, in a vector Y of length n, thanks to a logistic model, where the response Y is generated as a Bernoulli random variable of parameter logit^{-1}(XB), the coefficients B are sparse. In addition, the covariate matrix X is composed of correlated blocks of predictors.

Usage

```
sample.bin(n, p, kstar, lstar, beta.min, beta.max, mean.H = 0, sigma.H,
  sigma.F, seed = NULL)
```

Arguments

`n`	the number of observations in the sample.
`p`	the number of covariates in the sample.
`kstar`	the number of underlying latent variables used to generates the covariate matrix `X`, `kstar` <= `p`. `kstar` is also the number of blocks in the covariate matrix (see details).
`lstar`	the number of blocks in the covariate matrix `X` that are used to generates the response `Y`, i.e. with non null coefficients in vector `B`, `lstar` <= `kstar`.
`beta.min`	the inf bound for non null coefficients (see details).
`beta.max`	the sup bound for non null coefficients (see details).
`mean.H`	the mean of latent variables used to generates `X`.

sigma.H	the standard deviation of latent variables used to generates X.
sigma.F	the standard deviation of the noise added to latent variables used to generates X.
seed	an positive integer, if non NULL it fix the seed (with the command set.seed) used for random number generation.

Details

The set (1:p) of predictors is partitioned into kstar block. Each block k (k=1,...,kstar) depends on a latent variable H.k which are independent and identically distributed following a Gaussian distribution N(mean.H, sigma.H^2). Each columns X.j of the matrix X is generated as H.k + F.j for j in the block k, where F.j is independent and identically distributed gaussian noise N(0,sigma.F^2).

The coefficients B are generated as random between beta.min and beta.max on lstar blocks, randomly chosen, and null otherwise. The variables with non null coefficients are then relevant to explain the response, whereas the ones with null coefficients are not.

The response is generated as by drawing one observation of n different Bernoulli random variables of parameters logit^{-1}(XB).

The details of the procedure are developped by Durif et al. (2017).

Value

An object with the following attributes:

X	the (n x p) covariate matrix, containing the n observations for each of the p predictors.
Y	the (n) vector of Y observations.
proba	the n vector of Bernoulli parameters used to generate the response, in particular logit^{-1}(X %*% B).
sel	the index in (1:p) of covariates with non null coefficients in B.
nosel	the index in (1:p) of covariates with null coefficients in B.
B	the (n) vector of coefficients.
block.partition	a (p) vector indicating the block of each predictors in (1:kstar).
p	the number of covariates in the sample.
kstar	the number of underlying latent variables used to generates the covariate matrix X, kstar <= p. kstar is also the number of blocks in the covariate matrix (see details).
lstar	the number of blocks in the covariate matrix X that are used to generates the response Y, i.e. with non null coefficients in vector B, lstar <= kstar.
p0	the number of predictors with non null coefficients in B.
block.sel	a (lstar) vector indicating the index in (1:kstar) of blocks with predictors having non null coefficient in B.
beta.min	the inf bound for non null coefficients (see details).
beta.max	the sup bound for non null coefficients (see details).

mean.H	the mean of latent variables used to generates X.
sigma.H	the standard deviation of latent variables used to generates X.
sigma.F	the standard deviation of the noise added to latent variables used to generates X.
seed	an positive integer, if non NULL it fix the seed (with the command set.seed) used for random number generation.

Author(s)

Ghislain Durif (http://thoth.inrialpes.fr/people/gdurif/).

References

Durif G., Modolo L., Michaelsson J., Mold J. E., Lambert-Lacroix S., Picard F. (2017). High Dimensional Classification with combined Adaptive Sparse PLS and Logistic Regression, (in prep), available on (http://arxiv.org/abs/1502.05933).

See Also

sample.cont

Examples

```
### load plsgenomics library
library(plsgenomics)

### generating data
n <- 100
p <- 1000
sample1 <- sample.bin(n=n, p=p, kstar=20, lstar=2, beta.min=0.25,
                      beta.max=0.75, mean.H=0.2,
                      sigma.H=10, sigma.F=5)

str(sample1)
```

---

sample.cont

Generates design matrix X with correlated block of covariates and a continuous random reponse Y depening on X through gaussian linear model Y=XB+E

Description

The function sample.cont generates a random sample with p predictors X, a response Y, and n observations, through a linear model Y=XB+E, where the noise E is gaussian, the coefficients B are sparse, and the design matrix X is composed of correlated blocks of predictors.

Usage

```
sample.cont(n, p, kstar, lstar, beta.min, beta.max, mean.H=0, sigma.H,
               sigma.F, sigma.E, seed=NULL)
```

Arguments

n	the number of observations in the sample.
p	the number of covariates in the sample.
kstar	the number of underlying latent variables used to generates the design matrix X, kstar <= p. kstar is also the number of blocks in the design matrix (see details).
lstar	the number of blocks in the design matrix X that are used to generates the response Y, i.e. with non null coefficients in vector B, lstar <= kstar.
beta.min	the inf bound for non null coefficients (see details).
beta.max	the sup bound for non null coefficients (see details).
mean.H	the mean of latent variables used to generates X.
sigma.H	the standard deviation of latent variables used to generates X.
sigma.F	the standard deviation of the noise added to latent variables used to generates X.
sigma.E	the standard deviation of the noise in the linear model Y = X %*%* B + E used to generates Y.
seed	an positive integer, if non NULL it fix the seed (with the command set.seed) used for random number generation.

Details

The set (1:p) of predictors is partitioned into kstar block. Each block k (k=1,...,kstar) depends on a latent variable H.k which are independent and identically distributed following a distribution N(mean.H, sigma.H^2). Each columns X.j of the matrix X is generated as H.k + F.j for j in the block k, where F.j is independent and identically distributed gaussian noise N(0,sigma.F^2).

The coefficients B are generated as random between beta.min and beta.max on lstar blocks, randomly chosen, and null otherwise. The variables with non null coefficients are then relevant to explain the response, whereas the ones with null coefficients are not.

The response is generated as Y = X %*% B + E, where E is some gaussian noise N(0,sigma.E^2).

The details of the procedure are developped by Durif et al. (2015).

Value

A list with the following components:

X	the (n x p) design matrix, containing the n observations for each of the p predictors.
Y	the (n) vector of Y observations.
residuals	the (n) vector corresponding to the noise E in the model Y = X %*% B + E.
sel	the index in (1:p) of covariates with non null coefficients in B.

nosel	the index in (1:p) of covariates with null coefficients in B.
B	the (n) vector of coefficients.
block.partition	a (p) vector indicating the block of each predictors in (1:kstar).
p	the number of covariates in the sample.
kstar	the number of underlying latent variables used to generates the design matrix X, kstar <= p. kstar is also the number of blocks in the design matrix (see details).
lstar	the number of blocks in the design matrix X that are used to generates the response Y, i.e. with non null coefficients in vector B, lstar <= kstar.
p0	the number of predictors with non null coefficients in B.
block.sel	a (lstar) vector indicating the index in (1:kstar) of blocks with predictors having non null coefficient in B.
beta.min	the inf bound for non null coefficients (see details).
beta.max	the sup bound for non null coefficients (see details).
mean.H	the mean of latent variables used to generates X.
sigma.H	the standard deviation of latent variables used to generates X.
sigma.F	the standard deviation of the noise added to latent variables used to generates X.
sigma.E	the standard deviation of the noise in the linear model.
seed	an positive integer, if non NULL it fix the seed (with the command set.seed) used for random number generation.

Author(s)

Ghislain Durif (http://lbbe.univ-lyon1.fr/-Durif-Ghislain-.html).

References

G. Durif, F. Picard, S. Lambert-Lacroix (2015). Adaptive sparse PLS for logistic regression, (in prep), available on (http://arxiv.org/).

See Also

sample.bin.

Examples

```
### load plsgenomics library
library(plsgenomics)

### generating data
n <- 100
p <- 1000
sample1 <- sample.cont(n=n, p=p, kstar=20, lstar=2, beta.min=0.25, beta.max=0.75, mean.H=0.2,
sigma.H=10, sigma.F=5, sigma.E=5)
str(sample1)
```

`sample.multinom` *Generates covariate matrix X with correlated block of covariates and a multi-label random reponse depening on X through a multinomial model*

Description

The function `sample.multinom` generates a random sample of n observations, composed of p predictors, collected in the n x p matrix X, and a binary response, in a vector Y of length n, thanks to a logistic model, where the response Y is generated as a Bernoulli random variable of parameter logit^{-1}(XB), the coefficients B are sparse. In addition, the covariate matrix X is composed of correlated blocks of predictors.

Usage

```
sample.multinom(n, p, nb.class = 2, kstar, lstar, beta.min, beta.max,
  mean.H = 0, sigma.H, sigma.F, seed = NULL)
```

Arguments

`n`	the number of observations in the sample.
`p`	the number of covariates in the sample.
`nb.class`	the number of groups in the data.
`kstar`	the number of underlying latent variables used to generates the covariate matrix `X`, `kstar` <= `p`. `kstar` is also the number of blocks in the covariate matrix (see details).
`lstar`	the number of blocks in the covariate matrix `X` that are used to generates the response `Y`, i.e. with non null coefficients in vector `B`, `lstar` <= `kstar`.
`beta.min`	the inf bound for non null coefficients (see details).
`beta.max`	the sup bound for non null coefficients (see details).
`mean.H`	the mean of latent variables used to generates `X`.
`sigma.H`	the standard deviation of latent variables used to generates `X`.
`sigma.F`	the standard deviation of the noise added to latent variables used to generates `X`.
`seed`	an positive integer, if non NULL it fix the seed (with the command `set.seed`) used for random number generation.

Details

The set (1:p) of predictors is partitioned into kstar block. Each block k (k=1,...,kstar) depends on a latent variable H.k which are independent and identically distributed following a Gaussian distribution N(mean.H, sigma.H^2). Each columns X.j of the matrix X is generated as H.k + F.j for j in the block k, where F.j is independent and identically distributed gaussian noise N(0,sigma.F^2).

The coefficients B are generated as random between beta.min and beta.max on lstar blocks, randomly chosen, and null otherwise. The variables with non null coefficients are then relevant to explain the response, whereas the ones with null coefficients are not.

The response is generated as by drawing one observation of n different Bernoulli random variables of parameters logit^{-1}(XB).

The details of the procedure are developped by Durif et al. (2017).

Value

An object with the following attributes:

`X`	the (n x p) covariate matrix, containing the `n` observations for each of the `p` predictors.
`Y`	the (n) vector of Y observations.
`proba`	the n vector of Bernoulli parameters used to generate the response, in particular `logit^{-1}(X %*% B)`.
`sel`	the index in (1:p) of covariates with non null coefficients in `B`.
`nosel`	the index in (1:p) of covariates with null coefficients in `B`.
`B`	the (n) vector of coefficients.
`block.partition`	a (p) vector indicating the block of each predictors in (1:kstar).
`p`	the number of covariates in the sample.
`kstar`	the number of underlying latent variables used to generates the covariate matrix `X`, `kstar` <= `p`. `kstar` is also the number of blocks in the covariate matrix (see details).
`lstar`	the number of blocks in the covariate matrix `X` that are used to generates the response `Y`, i.e. with non null coefficients in vector `B`, `lstar` <= `kstar`.
`p0`	the number of predictors with non null coefficients in `B`.
`block.sel`	a (lstar) vector indicating the index in (1:kstar) of blocks with predictors having non null coefficient in `B`.
`beta.min`	the inf bound for non null coefficients (see details).
`beta.max`	the sup bound for non null coefficients (see details).
`mean.H`	the mean of latent variables used to generates `X`.
`sigma.H`	the standard deviation of latent variables used to generates `X`.
`sigma.F`	the standard deviation of the noise added to latent variables used to generates `X`.
`seed`	an positive integer, if non NULL it fix the seed (with the command `set.seed`) used for random number generation.

Author(s)

Ghislain Durif (http://thoth.inrialpes.fr/people/gdurif/).

References

Durif G., Modolo L., Michaelsson J., Mold J. E., Lambert-Lacroix S., Picard F. (2017). High Dimensional Classification with combined Adaptive Sparse PLS and Logistic Regression, (in prep), available on (http://arxiv.org/abs/1502.05933).

See Also

sample.cont

Examples

```
### load plsgenomics library
library(plsgenomics)

### generating data
n <- 100
p <- 1000
nclass <- 3
sample1 <- sample.multinom(n=n, p=p, nb.class=nclass,
                           kstar=20, lstar=2, beta.min=0.25,
                           beta.max=0.75, mean.H=0.2,
                           sigma.H=10, sigma.F=5)

str(sample1)
```

---

spls *Adaptive Sparse Partial Least Squares (SPLS) regression*

Description

The function spls.adapt performs compression and variable selection in the context of linear regression (with possible prediction) using Durif et al. (2017) adaptive SPLS algorithm.

Usage

```
spls(Xtrain, Ytrain, lambda.l1, ncomp, weight.mat = NULL, Xtest = NULL,
  adapt = TRUE, center.X = TRUE, center.Y = TRUE, scale.X = TRUE,
  scale.Y = TRUE, weighted.center = FALSE)
```

Arguments

Xtrain	a (ntrain x p) data matrix of predictor values. Xtrain must be a matrix. Each row corresponds to an observation and each column to a predictor variable.
Ytrain	a (ntrain) vector of (continuous) responses. Ytrain must be a vector or a one column matrix, and contains the response variable for each observation.
lambda.l1	a positive real value, in [0,1]. lambda.l1 is the sparse penalty parameter for the dimension reduction step by sparse PLS (see details).
ncomp	a positive integer. ncomp is the number of PLS components.
weight.mat	a (ntrain x ntrain) matrix used to weight the l2 metric in the observation space, it can be the covariance inverse of the Ytrain observations in a heteroskedastic context. If NULL, the l2 metric is the standard one, corresponding to homoskedastic model (weight.mat is the identity matrix).

Xtest	a (ntest x p) matrix containing the predictor values for the test data set. Xtest may also be a vector of length p (corresponding to only one test observation). Default value is NULL, meaning that no prediction is performed.
adapt	a boolean value, indicating whether the sparse PLS selection step sould be adaptive or not (see details).
center.X	a boolean value indicating whether the data matrices Xtrain and Xtest (if provided) should be centered or not.
center.Y	a boolean value indicating whether the response values Ytrain set should be centered or not.
scale.X	a boolean value indicating whether the data matrices Xtrain and Xtest (if provided) should be scaled or not (scale.X=TRUE implies center.X=TRUE).
scale.Y	a boolean value indicating whether the response values Ytrain should be scaled or not (scale.Y=TRUE implies center.Y=TRUE).
weighted.center	a boolean value indicating whether the centering should take into account the weighted l2 metric or not (if TRUE, it requires that weighted.mat is non NULL).

Details

The columns of the data matrices Xtrain and Xtest may not be standardized, since standardizing can be performed by the function spls as a preliminary step.

The procedure described in Durif et al. (2017) is used to compute latent sparse components that are used in a regression model. In addition, when a matrix Xtest is supplied, the procedure predicts the response associated to these new values of the predictors.

Value

An object of class spls with the following attributes

Xtrain	the ntrain x p predictor matrix.
Ytrain	the response observations.
sXtrain	the centered if so and scaled if so predictor matrix.
sYtrain	the centered if so and scaled if so response.
betahat	the linear coefficients in model sYtrain = sXtrain %*% betahat + residuals.
betahat.nc	the (p+1) vector containing the coefficients and intercept for the non centered and non scaled model Ytrain = cbind(rep(1,ntrain),Xtrain) %*% betahat.nc + residuals.nc.
meanXtrain	the (p) vector of Xtrain column mean, used for centering if so.
sigmaXtrain	the (p) vector of Xtrain column standard deviation, used for scaling if so.
meanYtrain	the mean of Ytrain, used for centering if so.
sigmaYtrain	the standard deviation of Ytrain, used for centering if so.
X.score	a (n x ncomp) matrix being the observations coordinates or scores in the new component basis produced by the compression step (sparse PLS). Each column t.k of X.score is a SPLS component.

X.score.low	a (n x ncomp) matrix being the PLS components only computed with the selected predictors.
X.loading	the (ncomp x p) matrix of coefficients in regression of Xtrain over the new components X.score.
Y.loading	the (ncomp) vector of coefficients in regression of Ytrain over the SPLS components X.score.
X.weight	a (p x ncomp) matrix being the coefficients of predictors in each components produced by sparse PLS. Each column w.k of X.weight verifies t.k = Xtrain x w.k (as a matrix product).
residuals	the (ntrain) vector of residuals in the model sYtrain = sXtrain %*% betahat + residuals.
residuals.nc	the (ntrain) vector of residuals in the non centered and non scaled model Ytrain = cbind(rep(1,ntrain),Xtrain) %*% betahat.nc + residuals.nc.
hatY	the (ntrain) vector containing the estimated reponse values on the train set of centered and scaled (if so) predictors sXtrain, hatY = sXtrain %*% betahat.
hatY.nc	the (ntrain) vector containing the estimated reponse value on the train set of non centered and non scaled predictors Xtrain, hatY.nc = cbind(rep(1,ntrain),Xtrain) %*% betahat.nc.
hatYtest	the (ntest) vector containing the predicted values for the response on the centered and scaled test set sXtest (if provided), hatYtest = sXtest %*% betahat.
hatYtest.nc	the (ntest) vector containing the predicted values for the response on the non centered and non scaled test set Xtest (if provided), hatYtest.nc = cbind(rep(1,ntest),Xtest) %*% betahat.nc.
A	the active set of predictors selected by the procedures. A is a subset of 1:p.
betamat	a (ncomp) list of coefficient vector betahat in the model with k components, for k=1,...,ncomp.
new2As	a (ncomp) list of subset of (1:p) indicating the variables that are selected when constructing the components k, for k=1,...,ncomp.
lambda.l1	the sparse hyper-parameter used to fit the model.
ncomp	the number of components used to fit the model.
V	the (ntrain x ntrain) matrix used to weight the metric in the sparse PLS step.
adapt	a boolean value, indicating whether the sparse PLS selection step was adaptive or not.

Author(s)

Ghislain Durif (http://thoth.inrialpes.fr/people/gdurif/).

Adapted in part from spls code by H. Chun, D. Chung and S.Keles (https://CRAN.R-project.org/package=spls).

References

Durif G., Modolo L., Michaelsson J., Mold J. E., Lambert-Lacroix S., Picard F. (2017). High Dimensional Classification with combined Adaptive Sparse PLS and Logistic Regression, (in prep), available on (http://arxiv.org/abs/1502.05933).

Chun, H., & Keles, S. (2010). Sparse partial least squares regression for simultaneous dimension reduction and variable selection. Journal of the Royal Statistical Society. Series B (Methodological), 72(1), 3-25. doi:10.1111/j.1467-9868.2009.00723.x

See Also

spls.cv

Examples

```
### load plsgenomics library
library(plsgenomics)

### generating data
n <- 100
p <- 100
sample1 <- sample.cont(n=n, p=p, kstar=10, lstar=2, beta.min=0.25,
                       beta.max=0.75, mean.H=0.2, sigma.H=10,
                       sigma.F=5, sigma.E=5)
X <- sample1$X
Y <- sample1$Y
### splitting between learning and testing set
index.train <- sort(sample(1:n, size=round(0.7*n)))
index.test <- (1:n)[-index.train]
Xtrain <- X[index.train,]
Ytrain <- Y[index.train,]
Xtest <- X[index.test,]
Ytest <- Y[index.test,]

### fitting the model, and predicting new observations
model1 <- spls(Xtrain=Xtrain, Ytrain=Ytrain, lambda.l1=0.5, ncomp=2,
               weight.mat=NULL, Xtest=Xtest, adapt=TRUE, center.X=TRUE,
               center.Y=TRUE, scale.X=TRUE, scale.Y=TRUE,
               weighted.center=FALSE)

str(model1)

### plotting the estimation versus real values for the non centered response
plot(model1$Ytrain, model1$hatY.nc,
     xlab="real Ytrain", ylab="Ytrain estimates")
points(-1000:1000,-1000:1000, type="l")

### plotting residuals versus centered response values
plot(model1$sYtrain, model1$residuals, xlab="sYtrain", ylab="residuals")

### plotting the predictor coefficients
plot(model1$betahat.nc, xlab="variable index", ylab="coeff")
```

```
### mean squares error of prediction on test sample
sYtest <- as.matrix(scale(Ytest, center=model1$meanYtrain, scale=model1$sigmaYtrain))
sum((model1$hatYtest - sYtest)^2) / length(index.test)

### plotting predicted values versus non centered real response values
## on the test set
plot(model1$hatYtest, sYtest, xlab="real Ytest", ylab="predicted values")
points(-1000:1000,-1000:1000, type="l")
```

`spls.cv` *Cross-validation procedure to calibrate the parameters (ncomp, lambda.l1) of the Adaptive Sparse PLS regression*

Description

The function `spls.cv` chooses the optimal values for the hyper-parameter of the `spls` procedure, by minimizing the mean squared error of prediction over the hyper-parameter grid, using Durif et al. (2017) adaptive SPLS algorithm.

Usage

```
spls.cv(X, Y, lambda.l1.range, ncomp.range, weight.mat = NULL, adapt = TRUE,
  center.X = TRUE, center.Y = TRUE, scale.X = TRUE, scale.Y = TRUE,
  weighted.center = FALSE, return.grid = FALSE, ncores = 1, nfolds = 10,
  nrun = 1, verbose = FALSE)
```

Arguments

`X`	a (n x p) data matrix of predictors. `X` must be a matrix. Each row corresponds to an observation and each column to a predictor variable.
`Y`	a (n) vector of (continuous) responses. `Y` must be a vector or a one column matrix. It contains the response variable for each observation.
`lambda.l1.range`	a vecor of positive real values, in [0,1]. `lambda.l1` is the sparse penalty parameter for the dimension reduction step by sparse PLS (see details), the optimal value will be chosen among `lambda.l1.range`.
`ncomp.range`	a vector of positive integers. `ncomp` is the number of PLS components. The optimal value will be chosen among `ncomp.range`.
`weight.mat`	a (ntrain x ntrain) matrix used to weight the l2 metric in the observation space, it can be the covariance inverse of the Ytrain observations in a heteroskedastic context. If NULL, the l2 metric is the standard one, corresponding to homoskedastic model (`weight.mat` is the identity matrix).
`adapt`	a boolean value, indicating whether the sparse PLS selection step sould be adaptive or not (see details).

center.X	a boolean value indicating whether the data matrices Xtrain and Xtest (if provided) should be centered or not.
center.Y	a boolean value indicating whether the response values Ytrain set should be centered or not.
scale.Y	a boolean value indicating whether the response values Ytrain should be scaled or not (scale.Y=TRUE implies center.Y=TRUE).
weighted.center	a boolean value indicating whether the centering should take into account the weighted l2 metric or not (if TRUE, it requires that weighted.mat is non NULL).
return.grid	a boolean values indicating whether the grid of hyper-parameters values with corresponding mean prediction error rate over the folds should be returned or not.
ncores	a positve integer, indicating the number of cores that the cross-validation is allowed to use for parallel computation (see details).
nfolds	a positive integer indicating the number of folds in the K-folds cross-validation procedure, nfolds=n corresponds to the leave-one-out cross-validation, default is 10.
nrun	a positive integer indicating how many times the K-folds cross- validation procedure should be repeated, default is 1.
verbose	a boolean value indicating verbosity.
scale.X	aaboolean value indicating whether the data matrices Xtrain and Xtest (if provided) should be scaled or not (scale.X=TRUE implies center.X=TRUE).

Details

The columns of the data matrices Xtrain and Xtest may not be standardized, since standardizing can be performed by the function spls.cv as a preliminary step.

The procedure is described in Durif et al. (2017). The K-fold cross-validation can be summarize as follow: the train set is partitioned into K folds, for each value of hyper-parameters the model is fit K times, using each fold to compute the prediction error rate, and fitting the model on the remaining observations. The cross-validation procedure returns the optimal hyper-parameters values, meaning the one that minimize the mean squared error of prediction averaged over all the folds.

This procedures uses the mclapply from the parallel package, available on GNU/Linux and MacOS. Users of Microsoft Windows can refer to the README file in the source to be able to use a mclapply type function.

Value

An object with the following attributes

lambda.l1.opt	the optimal value in lambda.l1.range.
ncomp.opt	the optimal value in ncomp.range.
cv.grid	the grid of hyper-parameters and corresponding prediction error rate over the folds. cv.grid is NULL if return.grid is set to FALSE.

Author(s)

Ghislain Durif (http://thoth.inrialpes.fr/people/gdurif/).

References

Durif G., Modolo L., Michaelsson J., Mold J. E., Lambert-Lacroix S., Picard F. (2017). High Dimensional Classification with combined Adaptive Sparse PLS and Logistic Regression, (in prep), available on (http://arxiv.org/abs/1502.05933).

See Also

spls

Examples

```
## Not run:
### load plsgenomics library
library(plsgenomics)

### generating data
n <- 100
p <- 100
sample1 <- sample.cont(n=n, p=p, kstar=10, lstar=2,
                       beta.min=0.25, beta.max=0.75, mean.H=0.2,
                       sigma.H=10, sigma.F=5, sigma.E=5)

X <- sample1$X
Y <- sample1$Y

### hyper-parameters values to test
lambda.l1.range <- seq(0.05,0.95,by=0.1) # between 0 and 1
ncomp.range <- 1:10

### tuning the hyper-parameters
cv1 <- spls.cv(X=X, Y=Y, lambda.l1.range=lambda.l1.range,
               ncomp.range=ncomp.range, weight.mat=NULL, adapt=TRUE,
               center.X=TRUE, center.Y=TRUE,
               scale.X=TRUE, scale.Y=TRUE, weighted.center=FALSE,
               return.grid=TRUE, ncores=1, nfolds=10, nrun=1)
str(cv1)

### otpimal values
cv1$lambda.l1.opt
cv1$ncomp.opt

## End(Not run)
```

spls.stab *Stability selection procedure to estimate probabilities of selection of covariates for the sparse PLS method*

Description

The function spls.stab train a sparse PLS model for each candidate values (ncomp, lambda.l1) of hyper-parameters on multiple sub-samplings in the data. The stability selection procedure selects the covariates that are selected by most of the models among the grid of hyper-parameters, following the procedure described in Durif et al. (2017). Candidates values for ncomp and lambda.l1 are respectively given by the input arguments ncomp.range and lambda.l1.range.

Usage

```
spls.stab(X, Y, lambda.l1.range, ncomp.range, weight.mat = NULL,
  adapt = TRUE, center.X = TRUE, center.Y = TRUE, scale.X = TRUE,
  scale.Y = TRUE, weighted.center = FALSE, ncores = 1, nresamp = 100,
  seed = NULL, verbose = TRUE)
```

Arguments

X	a (n x p) data matrix of predictors. X must be a matrix. Each row corresponds to an observation and each column to a predictor variable.
Y	a (n) vector of (continuous) responses. Y must be a vector or a one column matrix. It contains the response variable for each observation. Y should take values in {0,1}.
lambda.l1.range	a vecor of positive real values, in [0,1]. lambda.l1 is the sparse penalty parameter for the dimension reduction step by sparse PLS (see details), the optimal value will be chosen among lambda.l1.range.
ncomp.range	a vector of positive integers. ncomp is the number of PLS components. The optimal value will be chosen among ncomp.range.
weight.mat	a (ntrain x ntrain) matrix used to weight the l2 metric in the observation space, it can be the covariance inverse of the Ytrain observations in a heteroskedastic context. If NULL, the l2 metric is the standard one, corresponding to homoskedastic model (weight.mat is the identity matrix).
adapt	a boolean value, indicating whether the sparse PLS selection step sould be adaptive or not (see details).
center.X	a boolean value indicating whether the data matrices Xtrain and Xtest (if provided) should be centered or not.
center.Y	a boolean value indicating whether the response values Ytrain set should be centered or not.
scale.Y	a boolean value indicating whether the response values Ytrain should be scaled or not (scale.Y=TRUE implies center.Y=TRUE).

`weighted.center`
: a boolean value indicating whether the centering should take into account the weighted l2 metric or not (if TRUE, it requires that weighted.mat is non NULL).

`ncores`
: a positve integer, indicating the number of cores that the cross-validation is allowed to use for parallel computation (see details).

`nresamp`
: number of resamplings of the data to estimate the probility of selection for each covariate, default is 100.

`seed`
: a positive integer value (default is NULL). If non NULL, the seed for pseudo-random number generation is set accordingly.

`verbose`
: a boolean parameter indicating the verbosity.

`scale.X`
: aaboolean value indicating whether the data matrices `Xtrain` and `Xtest` (if provided) should be scaled or not (`scale.X=TRUE` implies `center.X=TRUE`).

Details

The columns of the data matrices `X` may not be standardized, since standardizing is performed by the function `spls.stab` as a preliminary step.

The procedure is described in Durif et al. (2017). The stability selection procedure can be summarize as follow (c.f. Meinshausen and Buhlmann, 2010).

(i) For each candidate values `(ncomp, lambda.l1)` of hyper-parameters, a logit-SPLS is trained on `nresamp` resamplings of the data. Then, for each pair `(ncomp, lambda.l1)`, the probability that a covariate (i.e. a column in `X`) is selected is computed among the resamplings.

(ii) Eventually, the set of "stable selected" variables corresponds to the set of covariates that were selected by most of the training among the grid of hyper-parameters candidate values.

This function achieves the first step (i) of the stability selection procedure. The second step (ii) is achieved by the function `stability.selection`.

This procedures uses `mclapply` from the `parallel` package, available on GNU/Linux and MacOS. Users of Microsoft Windows can refer to the README file in the source to be able to use a mclapply type function.

Value

An object with the following attributes

`q.Lambda`
: A table with values of q.Lambda (c.f. Durif et al. (2017) for the notation), being the averaged number of covariates selected among the entire grid of hyper-parameters candidates values, for increasing size of hyper-parameter grid.

`probs.lambda`
: A table with estimated probability of selection for each covariates depending on the candidates values for hyper-parameters.

`p`
: An integer values indicating the number of covariates in the model.

Author(s)

Ghislain Durif (http://thoth.inrialpes.fr/people/gdurif/).

References

Durif G., Modolo L., Michaelsson J., Mold J. E., Lambert-Lacroix S., Picard F. (2017). High Dimensional Classification with combined Adaptive Sparse PLS and Logistic Regression, (in prep), available on (http://arxiv.org/abs/1502.05933).

Meinshausen, N., Buhlmann P. (2010). Stability Selection. Journal of the Royal Statistical Society: Series B (Statistical Methodology) 72, no. 4, 417-473.

See Also

spls, stability.selection, stability.selection.heatmap

Examples

```
## Not run:
### load plsgenomics library
library(plsgenomics)

### generating data
n <- 100
p <- 100
sample1 <- sample.cont(n=n, p=p, kstar=10, lstar=2,
                       beta.min=0.25, beta.max=0.75, mean.H=0.2,
                       sigma.H=10, sigma.F=5, sigma.E=5)

X <- sample1$X
Y <- sample1$Y

### hyper-parameters values to test
lambda.l1.range <- seq(0.05,0.95,by=0.1) # between 0 and 1
ncomp.range <- 1:10

### tuning the hyper-parameters
stab1 <- spls.stab(X=X, Y=Y, lambda.l1.range=lambda.l1.range,
                   ncomp.range=ncomp.range,
                   adapt=TRUE,
                   ncores=1, nresamp=100)

str(stab1)

### heatmap of estimated probabilities
stability.selection.heatmap(stab1)

### selected covariates
stability.selection(stab1, piThreshold=0.6, rhoError=10)

## End(Not run)
```

SRBCT *Gene expression data from Khan et al. (2001)*

Description

Gene expression data (2308 genes for 83 samples) from the microarray experiments of Small Round Blue Cell Tumors (SRBCT) of childhood cancer study of Khan et al. (2001).

Usage

```
data(SRBCT)
```

Details

This data set contains 83 samples with 2308 genes: 29 cases of Ewing sarcoma (EWS), coded 1, 11 cases of Burkitt lymphoma (BL), coded 2, 18 cases of neuroblastoma (NB), coded 3, 25 cases of rhabdomyosarcoma (RMS), coded 4. A total of 63 training samples and 25 test samples are provided in Khan et al. (2001). Five of the test set are non-SRBCT and are not considered here. The training sample indexes correspond to 1:65 and the test sample indexes (without non-SRBCT sample) correspond to 66:83.

Value

A list with the following elements:

`X`	a (88 x 2308) matrix giving the expression levels of 2308 genes for 88 SRBCT patients. Each row corresponds to a patient, each column to a gene.
`Y`	a numeric vector of length 88 giving the cancer class of each patient.
`gene.names`	a matrix containing the names of the 2308 genes for the gene expression matrix `X`. The two columns correspond to the gene 'Image.Id.' and 'Gene.Description', respectively.

Source

The data are described in Khan et al. (2001) and can be freely downloaded from http://research.nhgri.nih.gov/microarray/Supplement/.

References

Khan, J. and Wei, J. S. and Ringner, M. and Saal, L. H. and Ladanyi, M. and Westermann, F. and Berthold, F. and Schwab, M. and Antonescu, C. R. and Peterson, C. and Meltzer, P. S. (2001). Classification and diagnostic prediction of cancers using gene expression profiling and artificial neural networks, Nature Medecine, 7, 673–679.

Examples

```
# load plsgenomics library
library(plsgenomics)

# load data set
data(SRBCT)

# how many samples and how many genes ?
dim(SRBCT$X)

# how many samples of class 1, 2, 3 and 4, respectively ?
sum(SRBCT$Y==1)
sum(SRBCT$Y==2)
sum(SRBCT$Y==3)
sum(SRBCT$Y==4)
```

---

stability.selection *Stability selection procedure to select covariates for the sparse PLS, LOGIT-SPLS and multinomial-SPLS methods*

Description

The function `stability.selection` returns the list of selected covariates, when following the stability selection procedure described in Durif et al. (2017). In particular, it selects covariates that are selected by most of the sparse PLS, the logit-SPLS or the multinomial-SPLS models when exploring the grid of hyper-parameter candidate values.

Usage

```
stability.selection(stab.out, piThreshold = 0.9, rhoError = 10)
```

Arguments

`stab.out`	the output of the functions `spls.stab`, `logit.spls.stab` or `multinom.spls.stab`.
`piThreshold`	a value in (0,1], corresponding to the threshold probability used to select covariate (c.f. Durif et al., 2017).
`rhoError`	a positive value used to restrict the grid of hyper-parameter candidate values (c.f. Durif et al., 2017).

Details

The procedure is described in Durif et al. (2017). The stability selection procedure can be summarize as follow (c.f. Meinshausen and Buhlmann, 2010).

(i) For each candidate values of hyper-parameters, a model is trained on `nresamp` resamplings of the data. Then, for each candidate value of the hyper-parameters, the probability that a covariate (i.e. a column in `X`) is selected is computed among the resamplings.

The estimated probabilities can be visualized as a heatmap with the function `stability.selection.heatmap`.

(ii) Eventually, the set of "stable selected" variables corresponds to the set of covariates that were selected by most of the training among the grid of hyper-parameters candidate values, based on a threshold probability `piThreshold` and a restriction of the grid of hyper-parameters based on `rhoError` (c.f. Durif et al., 2017, for details).

This function achieves the second step (ii) of the stability selection procedure. The first step (i) is achieved by the functions `spls.stab`, `logit.spls.stab` or `multinom.spls.stab`.

Value

An object with the following attributes:

`selected.predictors`
: The list of the name of covariates that are selected.

`max.probs`
: The corresponding estimated probabilities of selection for each covariate, i.e. the maximal values on the reduced grid of hyper-parameters.

Author(s)

Ghislain Durif (http://thoth.inrialpes.fr/people/gdurif/).

See Also

`spls.stab`, `logit.spls.stab`, `multinom.spls.stab`, `stability.selection.heatmap`

Examples

```
## Not run:
### load plsgenomics library
library(plsgenomics)

### generating data
n <- 100
p <- 100
sample1 <- sample.cont(n=n, p=p, kstar=10, lstar=2,
                       beta.min=0.25, beta.max=0.75, mean.H=0.2,
                       sigma.H=10, sigma.F=5, sigma.E=5)

X <- sample1$X
Y <- sample1$Y

### hyper-parameters values to test
lambda.l1.range <- seq(0.05,0.95,by=0.1) # between 0 and 1
ncomp.range <- 1:10

### tuning the hyper-parameters
stab1 <- spls.stab(X=X, Y=Y, lambda.l1.range=lambda.l1.range,
                   ncomp.range=ncomp.range, weight.mat=NULL,
                   adapt=FALSE, center.X=TRUE, center.Y=TRUE,
                   scale.X=TRUE, scale.Y=TRUE, weighted.center=FALSE,
                   ncores=1, nresamp=100)
```

```
str(stab1)

### selected covariates
stability.selection(stab1, piThreshold=0.6, rhoError=10)

## End(Not run)
```

`stability.selection.heatmap`

Heatmap visualization of estimated probabilities of selection for each covariate

---

Description

The function `stability.selection.heatmap` allows to visualize estimated probabilities to be selected for each covariate depending on the value of hyper-parameters in the spls, logit-spls or multinomial-spls models. These estimated probabilities are used in the stability selection procedure described in Durif et al. (2017).

Usage

```
stability.selection.heatmap(stab.out, ...)
```

Arguments

`stab.out`	the output of the functions `spls.stab`, `logit.spls.stab` or `multinom.spls.stab`.
`...`	any argument that could be pass to the functions `image.plot` or `image`.

Details

The procedure is described in Durif et al. (2017). The stability selection procedure can be summarize as follow (c.f. Meinshausen and Buhlmann, 2010).

(i) For each candidate values of hyper-parameters, a model is trained on `nresamp` resamplings of the data. Then, for each candidate value of the hyper-parameters, the probability that a covariate (i.e. a column in `X`) is selected is computed among the resamplings.

The estimated probabilities can be visualized as a heatmap with the function `stability.selection.heatmap`.

(ii) Eventually, the set of "stable selected" variables corresponds to the set of covariates that were selected by most of the training among the grid of hyper-parameters candidate values, based on a threshold probability `piThreshold` and a restriction of the grid of hyper-parameters based on `rhoError` (c.f. Durif et al., 2017, for details).

This function allows to visualize probabalities estimated at the first step (i) of the stability selection by the functions `spls.stab`, `logit.spls.stab` or `multinom.spls.stab`.

This function use the function `matrix.heatmap`.

Value

No return, just plot the heatmap in the current graphic window.

Author(s)

Ghislain Durif (http://thoth.inrialpes.fr/people/gdurif/).

See Also

logit.spls, stability.selection, stability.selection.heatmap

Examples

```
## Not run:
### load plsgenomics library
library(plsgenomics)

### generating data
n <- 100
p <- 100
sample1 <- sample.cont(n=n, p=p, kstar=10, lstar=2,
                       beta.min=0.25, beta.max=0.75, mean.H=0.2,
                       sigma.H=10, sigma.F=5, sigma.E=5)

X <- sample1$X
Y <- sample1$Y

### hyper-parameters values to test
lambda.l1.range <- seq(0.05,0.95,by=0.1) # between 0 and 1
ncomp.range <- 1:10

### tuning the hyper-parameters
stab1 <- spls.stab(X=X, Y=Y, lambda.l1.range=lambda.l1.range,
                   ncomp.range=ncomp.range, weight.mat=NULL,
                   adapt=FALSE, center.X=TRUE, center.Y=TRUE,
                   scale.X=TRUE, scale.Y=TRUE, weighted.center=FALSE,
                   ncores=1, nresamp=100)

str(stab1)

### heatmap of estimated probabilities
stability.selection.heatmap(stab1)

## End(Not run)
```

`TFA.estimate` *Prediction of Transcription Factor Activities using PLS*

Description

The function `TFA.estimate` estimates the transcription factor activities from gene expression data and ChIP data using the PLS multivariate regression approach described in Boulesteix and Strimmer (2005).

Usage

```
TFA.estimate(CONNECdata, GEdata, ncomp=NULL, nruncv=0, alpha=2/3, unit.weights=TRUE)
```

Arguments

`CONNECdata`	a (n x p) matrix containing the ChIP data for the n genes and the p predictors. The n genes must be the same as the n genes of `GEdata` and the ordering of the genes must also be the same. Each row of `ChIPdata` corresponds to a gene, each column to a transcription factor. `CONNECdata` might have either binary (e.g. 0-1) or numeric entries.
`GEdata`	a (n x m) matrix containing the gene expression levels of the n considered genes for m samples. Each row of `GEdata` corresponds to a gene, each column to a sample.
`ncomp`	if `nruncv=0`, `ncomp` is the number of latent components to be constructed. If `nruncv>0`, the number of latent components to be used for PLS regression is chosen from 1,...,`ncomp` using the cross-validation procedure described in Boulesteix and Strimmer (2005). If `ncomp=NULL`, `ncomp` is set to min(n,p).
`nruncv`	the number of cross-validation iterations to be performed for the choice of the number of latent components. If `nruncv=0`, cross-validation is not performed and `ncomp` latent components are used.
`alpha`	the proportion of genes to be included in the training set for the cross-validation procedure.
`unit.weights`	If `TRUE` then the latent components will be constructed from weight vectors that are standardized to length 1, otherwise the weight vectors do not have length 1 but the latent components have norm 1.

Details

The gene expression data as well as the ChIP data are assumed to have been properly normalized. However, they do not have to be centered or scaled, since centering and scaling are performed by the function `TFA.estimate` as a preliminary step.

The matrix `ChIPdata` containing the ChIP data for the n genes and p transcription factors might be replaced by any 'connectivity' matrix whose entries give the strength of the interactions between the genes and transcription factors. For instance, a connectivity matrix obtained by aggregating qualitative information from various genomic databases might be used as argument in place of ChIP data.

Value

A list with the following components:

TFA	a (p x m) matrix containing the estimated transcription factor activities for the p transcription factors and the m samples.
metafactor	a (m x ncomp) matrix containing the metafactors for the m samples. Each row corresponds to a sample, each column to a metafactor.
ncomp	the number of latent components used in the PLS regression.

Author(s)

Anne-Laure Boulesteix (http://www.ibe.med.uni-muenchen.de/organisation/mitarbeiter/020_professuren/boulesteix/eng.html) and Korbinian Strimmer (http://strimmerlab.org).

References

A. L. Boulesteix and K. Strimmer (2005). Predicting Transcription Factor Activities from Combined Analysis of Microarray and ChIP Data: A Partial Least Squares Approach.

A. L. Boulesteix, K. Strimmer (2007). Partial least squares: a versatile tool for the analysis of high-dimensional genomic data. Briefings in Bioinformatics 7:32-44.

S. de Jong (1993). SIMPLS: an alternative approach to partial least squares regression, Chemometrics Intell. Lab. Syst. **18**, 251–263.

See Also

pls.regression, pls.regression.cv.

Examples

```
# load plsgenomics library
library(plsgenomics)

# load Ecoli data
data(Ecoli)

# estimate TFAs based on 3 latent components
TFA.estimate(Ecoli$CONNECdata,Ecoli$GEdata,ncomp=3,nruncv=0)

# estimate TFAs and determine the best number of latent components simultaneously
TFA.estimate(Ecoli$CONNECdata,Ecoli$GEdata,ncomp=1:5,nruncv=20)
```

`variable.selection` *Variable selection using the PLS weights*

Description

The function `variable.selection` performs variable selection for binary classification.

Usage

```
variable.selection(X, Y, nvar=NULL)
```

Arguments

`X`	a (n x p) data matrix of predictors. X may be a matrix or a data frame. Each row corresponds to an observation and each column corresponds to a predictor variable.
`Y`	a vector of length n giving the classes of the n observations. The two classes must be coded as 1,2.
`nvar`	the number of variables to be returned. If `nvar=NULL`, all the variables are returned.

Details

The function `variable.selection` orders the variables according to the absolute value of the weight defining the first PLS component. This ordering is equivalent to the ordering obtained with the F-statistic and t-test with equal variances (Boulesteix, 2004).

For computational reasons, the function `variable.selection` does not use the pls algorithm, but the obtained ordering of the variables is exactly equivalent to the ordering obtained using the PLS weights output by `pls.regression`.

Value

A vector of length `nvar` (or of length p if `nvar=NULL`) containing the indices of the variables to be selected. The variables are ordered from the best to the worst variable.

Author(s)

Anne-Laure Boulesteix (http://www.ibe.med.uni-muenchen.de/organisation/mitarbeiter/020_professuren/boulesteix/eng.html)

References

A. L. Boulesteix (2004). PLS dimension reduction for classification with microarray data, Statistical Applications in Genetics and Molecular Biology **3**, Issue 1, Article 33.

A. L. Boulesteix, K. Strimmer (2007). Partial least squares: a versatile tool for the analysis of high-dimensional genomic data. Briefings in Bioinformatics 7:32-44.

S. de Jong (1993). SIMPLS: an alternative approach to partial least squares regression, Chemometrics Intell. Lab. Syst. **18**, 251–263.

See Also

pls.regression.

Examples

```
# load plsgenomics library
library(plsgenomics)

# generate X and Y (4 observations and 3 variables)
X<-matrix(c(4,3,3,4,1,0,6,7,3,5,5,9),4,3,byrow=FALSE)
Y<-c(1,1,2,2)

# select the 2 best variables
variable.selection(X,Y,nvar=2)
# order the 3 variables
variable.selection(X,Y)

# load the leukemia data
data(leukemia)

# select the 50 best variables from the leukemia data
variable.selection(leukemia$X,leukemia$Y,nvar=50)
```

Index