Package 'sharp'

January 17, 2023

Type Package

Title Stability-enHanced Approaches using Resampling Procedures

Version 1.3.0

Date 2023-01-17

Author Barbara Bodinier [aut, cre]

Maintainer Barbara Bodinier <b.bodinier@imperial.ac.uk>

URL https://github.com/barbarabodinier/sharp

BugReports https://github.com/barbarabodinier/sharp/issues

Description In stability selection (N Meinshausen, P Bühlmann (2010) <doi:10.1111/j.1467-9868.2010.00740.x>) and consensus clustering (S Monti et al (2003) <doi:10.1023/A:1023949509487>), resampling techniques are used to enhance the reliability of the results. In this package, hyperparameters are calibrated by maximising model stability, which is measured by the negative log-likelihood under the null hypothesis that all selection (or co-membership) probabilities are identical (B Bodinier et al (2021) <arXiv:2106.02521>). Functions are readily implemented for the use of LASSO regression, sparse PCA, sparse (group) PLS or graphical LASSO in stability selection, and hierarchical clustering, partitioning around medoids, K means or Gaussian mixture models in consensus clustering.

License GPL (>= 3)

Language en-GB

Encoding UTF-8

RoxygenNote 7.2.0

Depends fake (>= 1.3.0), R (>= 3.5)

Imports beepr, glassoFast (>= 1.0.0), glmnet, grDevices, huge, igraph, impute, MASS, mclust, parallel, randomcoloR, Rdpack, withr (>= 2.4.0)

Suggests cluster, corpcor, dbscan, elasticnet, gglasso, mixOmics, nnet, plotrix, RCy3, rmarkdown, rCOSA, sgPLS, sparcl, survival (>= 3.2.13), testthat (>= 3.0.0), visNetwork

Additional_repositories https://barbarabodinier.github.io/drat

Config/testthat/edition 3

RdMacros Rdpack

NeedsCompilation no

Repository CRAN

Date/Publication 2023-01-17 22:50:02 UTC

R topics documented:

`sharp-package` *sharp: Stability-enHanced Approaches using Resampling Procedures*

Description

In stability selection and consensus clustering, resampling techniques are used to enhance the reliability of the results. In this package, hyper-parameters are calibrated by maximising model stability, which is measured by the negative log-likelihood under the null hypothesis that all selection (or co-membership) probabilities are identical. Functions are readily implemented for the use of LASSO regression, sparse PCA, sparse (group) PLS or graphical LASSO in stability selection, and hierarchical clustering, partitioning around medoids, K means or Gaussian mixture models in consensus clustering.

Details

Package:	sharp
Type:	Package
Version:	1.3.0
Date:	2023-01-17
License:	GPL (>= 3)
Maintainer:	Barbara Bodinier <b.bodinier@imperial.ac.uk>

References

Bodinier B, Filippi S, Nost TH, Chiquet J, Chadeau-Hyam M (2021). "Automated calibration for stability selection in penalised regression and graphical models: a multi-OMICs network application exploring the molecular response to tobacco smoking." https://arxiv.org/abs/2106.02521.

Meinshausen N, Bühlmann P (2010). "Stability selection." *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, **72**(4), 417-473. doi: 10.1111/j.14679868.2010.00740.x.

Monti S, Tamayo P, Mesirov J, Golub T (2003). "Consensus Clustering: A Resampling-Based Method for Class Discovery and Visualization of Gene Expression Microarray Data." *Machine Learning*, **52**(1), 91–118. doi: 10.1023/A:1023949509487.

Examples

```
oldpar <- par(no.readonly = TRUE)
par(mar = c(5, 5, 5, 5))

## Regression models
# Data simulation
set.seed(1)
simul <- SimulateRegression(n = 100, pk = 50)

# Stability selection
stab <- VariableSelection(xdata = simul$xdata, ydata = simul$ydata)
CalibrationPlot(stab)
summary(stab)
SelectedVariables(stab)


## Graphical models
# Data simulation
set.seed(1)
simul <- SimulateGraphical(n = 100, pk = 20, topology = "scale-free")

# Stability selection
stab <- GraphicalModel(xdata = simul$data)
CalibrationPlot(stab)
summary(stab)
plot(stab)


## PCA models
# Data simulation
set.seed(1)
simul <- SimulateComponents(pk = c(5, 3, 4))
plot(simul)

# Stability selection
stab <- BiSelection(
  xdata = simul$data,
  ncomp = 3,
```

```
  implementation = SparsePCA
)
CalibrationPlot(stab)
summary(stab)
SelectedVariables(stab)


## PLS models
# Data simulation
set.seed(1)
simul <- SimulateRegression(n = 50, pk = c(10, 20, 30), family = "gaussian")

# Stability selection
stab <- BiSelection(
  xdata = simul$xdata, ydata = simul$ydata,
  family = "gaussian", ncomp = 3,
  implementation = SparsePLS
)
CalibrationPlot(stab)
summary(stab)
plot(stab)

par(oldpar)
```

AggregatedEffects *Summarised coefficients conditionally on selection*

Description

Computes descriptive statistics (defined by FUN) for coefficients of the (calibrated) models conditionally on selection across resampling iterations.

Usage

```
AggregatedEffects(
  stability,
  lambda_id = NULL,
  side = "X",
  comp = 1,
  FUN = stats::median,
  ...
)
```

Arguments

stability	output of VariableSelection or BiSelection.
lambda_id	parameter ID with respect to the grid Lambda. If NULL, aggregated coefficients across the models run with the calibrated parameter are returned.

side	character string indicating if coefficients of predictors (side="X") or outcomes (side="Y") should be returned. Only applicable to PLS models.
comp	component ID. Only applicable to PLS models.
FUN	function to use to aggregate coefficients of visited models over resampling iterations. Recommended functions include median or mean.
...	additional arguments to be passed to FUN.

Value

A matrix of summarised coefficients conditionally on selection across resampling iterations. Missing values (NA) are returned for variables that are never selected.

See Also

VariableSelection, BiSelection, Refit

Examples

```
# Example with univariate outcome
set.seed(1)
simul <- SimulateRegression(n = 100, pk = 50, family = "gaussian")
stab <- VariableSelection(xdata = simul$xdata, ydata = simul$ydata, family = "gaussian")
median_betas <- AggregatedEffects(stab)

# Comparison with refitted model
refitted <- Refit(xdata = simul$xdata, ydata = simul$ydata, stability = stab)
refitted_betas <- refitted$coefficients[-1]
plot(median_betas[names(refitted_betas), ], refitted_betas,
  panel.first = abline(0, 1, lty = 2)
)

# Extracting mean betas conditionally on selection
mean_betas <- AggregatedEffects(stab, FUN = mean)
plot(median_betas, mean_betas)

# Regression with multivariate outcomes
set.seed(1)
simul <- SimulateRegression(n = 100, pk = 50, q = 2, family = "gaussian")
stab <- VariableSelection(xdata = simul$xdata, ydata = simul$ydata, family = "mgaussian")
median_betas <- AggregatedEffects(stab)
dim(median_betas)

# Sparse PLS with multivariate outcome
set.seed(1)
simul <- SimulateRegression(n = 50, pk = 15, q = 3, family = "gaussian")
x <- simul$xdata
y <- simul$ydata
stab <- BiSelection(
  xdata = x, ydata = y,
  family = "gaussian", ncomp = 3,
```

```
  LambdaX = 1:(ncol(x) - 1),
  implementation = SparsePLS
)
median_betas <- AggregatedEffects(stab)
dim(median_betas)
median_betas <- AggregatedEffects(stab, side = "Y")
dim(median_betas)
```

ArgmaxId *Calibrated hyper-parameter(s)*

Description

Extracts the calibrated hyper-parameters (or their indices for ArgmaxId) with respect to the grids provided in Lambda and pi_list in argument stability.

Usage

```
ArgmaxId(stability = NULL, S = NULL)

Argmax(stability)
```

Arguments

stability	output of VariableSelection or GraphicalModel.
S	matrix of stability scores obtained with different combinations of parameters where rows correspond to different values of the parameter controlling the level of sparsity in the underlying feature selection algorithm and columns correspond to different values of the threshold in selection proportions. If S=NULL, argument stability must be provided.

Value

A matrix of hyper-parameters (Argmax) or indices (ArgmaxId). For multi-block graphical models, rows correspond to different blocks.

See Also

VariableSelection, GraphicalModel

Examples

```
# Data simulation
set.seed(1)
simul <- SimulateGraphical(pk = 20)

# Stability selection
stab <- GraphicalModel(xdata = simul$data)

# Extracting calibrated hyper-parameters
Argmax(stab)

# Extracting calibrated hyper-parameters IDs
ids <- ArgmaxId(stab)
ids

# Relationship between the two functions
stab$Lambda[ids[1], 1]
stab$params$pi_list[ids[2]]
```

BiSelection *Stability selection of predictors and/or outcomes*

Description

Performs stability selection for dimensionality reduction. The underlying variable selection algorithm (e.g. sparse PLS) is run with different combinations of parameters controlling the sparsity (e.g. number of selected variables per component) and thresholds in selection proportions. These hyper-parameters are jointly calibrated by maximisation of the stability score.

Usage

```
BiSelection(
  xdata,
  ydata = NULL,
  group_x = NULL,
  group_y = NULL,
  LambdaX = NULL,
  LambdaY = NULL,
  AlphaX = NULL,
  AlphaY = NULL,
  ncomp = 1,
  scale = TRUE,
  pi_list = seq(0.6, 0.9, by = 0.01),
  K = 100,
  tau = 0.5,
  seed = 1,
  n_cat = 3,
```

```
  family = "gaussian",
  implementation = SparsePLS,
  resampling = "subsampling",
  cpss = FALSE,
  PFER_method = "MB",
  PFER_thr = Inf,
  FDP_thr = Inf,
  n_cores = 1,
  output_data = FALSE,
  verbose = TRUE,
  beep = NULL,
  ...
)
```

Arguments

xdata	matrix of predictors with observations as rows and variables as columns.
ydata	optional vector or matrix of outcome(s). If family is set to "binomial" or "multinomial", ydata can be a vector with character/numeric values or a factor.
group_x	vector encoding the grouping structure among predictors. This argument indicates the number of variables in each group. Only used for models with group penalisation (e.g. implementation=GroupPLS or implementation=SparseGroupPLS).
group_y	optional vector encoding the grouping structure among outcomes. This argument indicates the number of variables in each group. Only used if implementation=GroupPLS or implementation=SparseGroupPLS.
LambdaX	matrix of parameters controlling the number of selected variables (for sparse PCA/PLS) or groups (for group and sparse group PLS) in X.
LambdaY	matrix of parameters controlling the number of selected variables (for sparse PLS) or groups (for group or sparse group PLS) in Y. Only used if family="gaussian".
AlphaX	matrix of parameters controlling the level of sparsity within groups in X. Only used if implementation=SparseGroupPLS.
AlphaY	matrix of parameters controlling the level of sparsity within groups in X. Only used if implementation=SparseGroupPLS and family="gaussian".
ncomp	number of components.
scale	logical indicating if the data should be scaled (i.e. transformed so that all variables have a standard deviation of one).
pi_list	vector of thresholds in selection proportions. If n_cat=3, these values must be >0.5 and <1. If n_cat=2, these values must be >0 and <1.
K	number of resampling iterations.
tau	subsample size. Only used if resampling="subsampling" and cpss=FALSE.
seed	value of the seed to initialise the random number generator and ensure reproducibility of the results (see set.seed).
n_cat	number of categories used to compute the stability score. Possible values are 2 or 3.

family	type of PLS model. This parameter must be set to family="gaussian" for continuous outcomes, or to family="binomial" for categorical outcomes. Only used if ydata is provided.
implementation	function to use for feature selection. Possible functions are: SparsePCA, SparsePLS, GroupPLS, SparseGroupPLS.
resampling	resampling approach. Possible values are: "subsampling" for sampling without replacement of a proportion tau of the observations, or "bootstrap" for sampling with replacement generating a resampled dataset with as many observations as in the full sample. Alternatively, this argument can be a function to use for resampling. This function must use arguments named data and tau and return the IDs of observations to be included in the resampled dataset.
cpss	logical indicating if complementary pair stability selection should be done. For this, the algorithm is applied on two non-overlapping subsets of half of the observations. A feature is considered as selected if it is selected for both subsamples. With this method, the data is split K/2 times (K models are fitted). Only used if PFER_method="MB".
PFER_method	method used to compute the upper-bound of the expected number of False Positives (or Per Family Error Rate, PFER). If PFER_method="MB", the method proposed by Meinshausen and Bühlmann (2010) is used. If PFER_method="SS", the method proposed by Shah and Samworth (2013) under the assumption of unimodality is used.
PFER_thr	threshold in PFER for constrained calibration by error control. If PFER_thr=Inf and FDP_thr=Inf, unconstrained calibration is used (the default).
FDP_thr	threshold in the expected proportion of falsely selected features (or False Discovery Proportion) for constrained calibration by error control. If PFER_thr=Inf and FDP_thr=Inf, unconstrained calibration is used (the default).
n_cores	number of cores to use for parallel computing (see mclapply). Only available on Unix systems.
output_data	logical indicating if the input datasets xdata and ydata should be included in the output.
verbose	logical indicating if a loading bar and messages should be printed.
beep	sound indicating the end of the run. Possible values are: NULL (no sound) or an integer between 1 and 11 (see argument sound in beep).
...	additional parameters passed to the functions provided in implementation or resampling.

Details

In stability selection, a feature selection algorithm is fitted on K subsamples (or bootstrap samples) of the data with different parameters controlling the sparsity (LambdaX, LambdaY, AlphaX, and/or AlphaY). For a given (set of) sparsity parameter(s), the proportion out of the K models in which each feature is selected is calculated. Features with selection proportions above a threshold pi are considered stably selected. The stability selection model is controlled by the sparsity parameter(s) (denoted by λ) for the underlying algorithm, and the threshold in selection proportion:

$V_{\lambda,\pi} = \{j : p_\lambda(j) \geq \pi\}$

For sparse and sparse group dimensionality reduction, "feature" refers to variable (variable selection model). For group PLS, "feature" refers to group (group selection model). For (sparse) group PLS, groups need to be defined *a priori* and specified in arguments `group_x` and/or `group_y`.

These parameters can be calibrated by maximisation of a stability score (see `StabilityScore`) derived from the likelihood under the assumption of uniform (uninformative) selection:

$S_{\lambda,\pi} = -log(L_{\lambda,\pi})$

It is strongly recommended to examine the calibration plot carefully to check that the grids of parameters `Lambda` and `pi_list` do not restrict the calibration to a region that would not include the global maximum (see `CalibrationPlot`). In particular, the grid `Lambda` may need to be extended when the maximum stability is observed on the left or right edges of the calibration plot.

To control the expected number of False Positives (Per Family Error Rate) in the results, a threshold `PFER_thr` can be specified. The optimisation problem is then constrained to sets of parameters that generate models with an upper-bound in PFER below `PFER_thr` (see Meinshausen and Bühlmann (2010) and Shah and Samworth (2013)).

Possible resampling procedures include defining (i) `K` subsamples of a proportion `tau` of the observations, (ii) `K` bootstrap samples with the full sample size (obtained with replacement), and (iii) `K/2` splits of the data in half for complementary pair stability selection (see arguments `resampling` and `cpss`). In complementary pair stability selection, a feature is considered selected at a given resampling iteration if it is selected in the two complementary subsamples.

For categorical outcomes (argument `family` is `"binomial"` or `"multinomial"`), the proportions of observations from each category in all subsamples or bootstrap samples are the same as in the full sample.

To ensure reproducibility of the results, the starting number of the random number generator is set to `seed`.

For parallelisation, stability selection with different sets of parameters can be run on `n_cores` cores. This relies on forking with `mclapply` (specific to Unix systems).

Value

An object of class `bi_selection`. A list with:

`summary` a matrix of the best stability scores and corresponding parameters controlling the level of sparsity in the underlying algorithm for different numbers of components. Possible columns include: `comp` (component index), `nx` (number of predictors to include, parameter of the underlying algorithm), `alphax` (sparsity within the predictor groups, parameter of the underlying algorithm), `pix` (threshold in selection proportion for predictors), `ny` (number of outcomes to include, parameter of the underlying algorithm), `alphay` (sparsity within the outcome groups, parameter of the underlying algorithm), `piy` (threshold in selection proportion for outcomes), `S` (stability score). Columns that are not relevant to the model are not reported (e.g. `alpha_x` and `alpha_y` are not returned for sparse PLS models).

`summary_full` a matrix of the best stability scores for different combinations of parameters controlling the sparsity and components.

`selectedX` a binary matrix encoding stably selected predictors.

`selpropX` a matrix of calibrated selection proportions for predictors.

selectedY	a binary matrix encoding stably selected outcomes. Only returned for PLS models.
selpropY	a matrix of calibrated selection proportions for outcomes. Only returned for PLS models.
selected	a binary matrix encoding stable relationships between predictor and outcome variables. Only returned for PLS models.
selectedX_full	a binary matrix encoding stably selected predictors.
selpropX_full	a matrix of selection proportions for predictors.
selectedY_full	a binary matrix encoding stably selected outcomes. Only returned for PLS models.
selpropY_full	a matrix of selection proportions for outcomes. Only returned for PLS models.
coefX	an array of estimated loadings coefficients for the different components (rows), for the predictors (columns), as obtained across the K visited models (along the third dimension).
coefY	an array of estimated loadings coefficients for the different components (rows), for the outcomes (columns), as obtained across the K visited models (along the third dimension). Only returned for PLS models.
method	a list with type="bi_selection" and values used for arguments implementation, family, scale, resampling, cpss and PFER_method.
params	a list with values used for arguments K, group_x, group_y, LambdaX, LambdaY, AlphaX, AlphaY, pi_list, tau, n_cat, pk, n (number of observations), PFER_thr, FDP_thr and seed. The datasets xdata and ydata are also included if output_data=TRUE.

The rows of summary and columns of selectedX, selectedY, selpropX, selpropY, selected, coefX and coefY are ordered in the same way and correspond to components and parameter values stored in summary. The rows of summary_full and columns of selectedX_full, selectedY_full, selpropX_full and selpropY_full are ordered in the same way and correspond to components and parameter values stored in summary_full.

References

Bodinier B, Filippi S, Nost TH, Chiquet J, Chadeau-Hyam M (2021). “Automated calibration for stability selection in penalised regression and graphical models: a multi-OMICs network application exploring the molecular response to tobacco smoking.” https://arxiv.org/abs/2106.02521.

Shah RD, Samworth RJ (2013). “Variable selection with error control: another look at stability selection.” *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, **75**(1), 55-80. doi: 10.1111/j.14679868.2011.01034.x.

Meinshausen N, Bühlmann P (2010). “Stability selection.” *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, **72**(4), 417-473. doi: 10.1111/j.14679868.2010.00740.x.

Liquet B, de Micheaux PL, Hejblum BP, Thiébaut R (2016). “Group and sparse group partial least square approaches applied in genomics context.” *Bioinformatics*, **32**(1), 35-42. ISSN 1367-4803, doi: 10.1093/bioinformatics/btv535.

KA LC, Rossouw D, Robert-Granié C, Besse P (2008). “A sparse PLS for variable selection when integrating omics data.” *Stat Appl Genet Mol Biol*, **7**(1), Article 35. ISSN 1544-6115, doi: 10.2202/15446115.1390.

Shen H, Huang JZ (2008). "Sparse principal component analysis via regularized low rank matrix approximation." *Journal of Multivariate Analysis*, **99**(6), 1015-1034. ISSN 0047-259X, doi: 10.1016/j.jmva.2007.06.007.

Zou H, Hastie T, Tibshirani R (2006). "Sparse Principal Component Analysis." *Journal of Computational and Graphical Statistics*, **15**(2), 265-286. doi: 10.1198/106186006X113430.

See Also

SparsePCA, SparsePLS, GroupPLS, SparseGroupPLS, VariableSelection, Resample, StabilityScore

Other stability functions: Clustering(), GraphicalModel(), VariableSelection()

Examples

```
oldpar <- par(no.readonly = TRUE)
par(mar = c(12, 5, 1, 1))

## Sparse Principal Component Analysis

# Data simulation
set.seed(1)
simul <- SimulateComponents(pk = c(5, 3, 4))

# sPCA: sparsity on X (unsupervised)
stab <- BiSelection(
  xdata = simul$data,
  ncomp = 2,
  LambdaX = 1:(ncol(simul$data) - 1),
  implementation = SparsePCA
)
print(stab)

# Calibration plot
CalibrationPlot(stab)

# Visualisation of the results
summary(stab)
plot(stab)
SelectedVariables(stab)


## Sparse (Group) Partial Least Squares

# Data simulation (continuous outcomes)
set.seed(1)
simul <- SimulateRegression(n = 100, pk = 15, q = 3, family = "gaussian")
x <- simul$xdata
y <- simul$ydata

# sPLS: sparsity on X
stab <- BiSelection(
  xdata = x, ydata = y,
```

```
  family = "gaussian", ncomp = 3,
  LambdaX = 1:(ncol(x) - 1),
  implementation = SparsePLS
)
CalibrationPlot(stab)
summary(stab)
plot(stab)

# sPLS: sparsity on both X and Y
stab <- BiSelection(
  xdata = x, ydata = y,
  family = "gaussian", ncomp = 3,
  LambdaX = 1:(ncol(x) - 1),
  LambdaY = 1:(ncol(y) - 1),
  implementation = SparsePLS,
  n_cat = 2
)
CalibrationPlot(stab)
summary(stab)
plot(stab)

# sgPLS: sparsity on X
stab <- BiSelection(
  xdata = x, ydata = y, K = 10,
  group_x = c(2, 8, 5),
  family = "gaussian", ncomp = 3,
  LambdaX = 1:2, AlphaX = seq(0.1, 0.9, by = 0.1),
  implementation = SparseGroupPLS
)
CalibrationPlot(stab)
summary(stab)

par(oldpar)
```

BlockLambdaGrid *Multi-block grid*

Description

Generates a matrix of parameters controlling the sparsity of the underlying selection algorithm for multi-block calibration.

Usage

```
BlockLambdaGrid(Lambda, lambda_other_blocks = NULL)
```

Arguments

Lambda
: vector or matrix of penalty parameters.

lambda_other_blocks
: optional vector of penalty parameters to use for other blocks in the iterative multi-block procedure.

Value

A list with:

Lambda
: a matrix of (block-specific) penalty parameters. In multi-block stability selection, rows correspond to sets of penalty parameters and columns correspond to different blocks.

Sequential_template
: logical matrix encoding the type of procedure for data with multiple blocks in stability selection graphical modelling. For multi-block estimation, each block is calibrated separately while others blocks are weakly penalised (TRUE only for the block currently being calibrated and FALSE for other blocks). Other approaches with joint calibration of the blocks are allowed (all entries are set to TRUE).

See Also

GraphicalModel

Examples

```
# Multi-block grid
Lambda <- matrix(c(
  0.8, 0.6, 0.3,
  0.5, 0.4, 0.2,
  0.7, 0.5, 0.1
),
ncol = 3, byrow = TRUE
)
mygrid <- BlockLambdaGrid(Lambda, lambda_other_blocks = 0.1)

# Multi-parameter grid (not recommended)
Lambda <- matrix(c(
  0.8, 0.6, 0.3,
  0.5, 0.4, 0.2,
  0.7, 0.5, 0.1
),
ncol = 3, byrow = TRUE
)
mygrid <- BlockLambdaGrid(Lambda, lambda_other_blocks = NULL)
```

CalibrationPlot *Calibration plot*

Description

Creates a plot showing the stability score as a function of the parameter(s) controlling the level of sparsity in the underlying feature selection algorithm and/or the threshold in selection proportions. See examples in VariableSelection, GraphicalModel, Clustering and BiSelection.

Usage

```
CalibrationPlot(
  stability,
  block_id = NULL,
  col = NULL,
  pch = 19,
  cex = 0.7,
  xlim = NULL,
  ylim = NULL,
  bty = "o",
  lines = TRUE,
  lty = 3,
  lwd = 2,
  show_argmax = TRUE,
  show_pix = FALSE,
  show_piy = FALSE,
  offset = 0.3,
  legend = TRUE,
  legend_length = NULL,
  legend_range = NULL,
  ncol = 1,
  xlab = NULL,
  ylab = NULL,
  zlab = expression(italic(q)),
  xlas = 2,
  ylas = NULL,
  zlas = 2,
  cex.lab = 1.5,
  cex.axis = 1,
  cex.legend = 1.2,
  xgrid = FALSE,
  ygrid = FALSE,
  params = c("ny", "alphay", "nx", "alphax")
)
```

Arguments

stability output of VariableSelection, GraphicalModel or BiSelection.

block_id	ID of the block to visualise. Only used for multi-block stability selection graphical models. If block_id=NULL, all blocks are represented in separate panels.
col	vector of colours.
pch	type of point, as in points.
cex	size of point.
xlim	displayed range along the x-axis. Only used if stability is the output of BiSelection.
ylim	displayed range along the y-axis. Only used if stability is the output of BiSelection.
bty	character string indicating if the box around the plot should be drawn. Possible values include: "o" (default, the box is drawn), or "n" (no box).
lines	logical indicating if the points should be linked by lines. Only used if stability is the output of BiSelection or Clustering.
lty	line type, as in par. Only used if stability is the output of BiSelection.
lwd	line width, as in par. Only used if stability is the output of BiSelection.
show_argmax	logical indicating if the calibrated parameter(s) should be indicated by lines.
show_pix	logical indicating if the calibrated threshold in selection proportion in X should be written for each point. Only used if stability is the output of BiSelection.
show_piy	logical indicating if the calibrated threshold in selection proportion in Y should be written for each point. Only used if stability is the output of BiSelection with penalisation of the outcomes.
offset	distance between the point and the text, as in text. Only used if show_pix=TRUE or show_piy=TRUE.
legend	logical indicating if the legend should be included.
legend_length	length of the colour bar. Only used if stability is the output of VariableSelection or GraphicalModel.
legend_range	range of the colour bar. Only used if stability is the output of VariableSelection or GraphicalModel.
ncol	integer indicating the number of columns in the legend.
xlab	label of the x-axis.
ylab	label of the y-axis.
zlab	label of the z-axis. Only used if stability is the output of VariableSelection or GraphicalModel.
xlas	orientation of labels on the x-axis, as las in par.
ylas	orientation of labels on the y-axis, as las in par.
zlas	orientation of labels on the z-axis, as las in par.
cex.lab	font size for labels.
cex.axis	font size for axes.
cex.legend	font size for text legend entries.
xgrid	logical indicating if a vertical grid should be drawn. Only used if stability is the output of BiSelection.

ygrid	logical indicating if a horizontal grid should be drawn. Only used if stability is the output of BiSelection.
params	vector of possible parameters if stability is of class bi_selection. The order of these parameters defines the order in which they are represented. Only used if stability is the output of BiSelection.

Value

A calibration plot.

See Also

VariableSelection, GraphicalModel, Clustering, BiSelection

Clustering *Consensus clustering*

Description

Performs consensus (weighted) clustering. The underlying algorithm (e.g. hierarchical clustering) is run with different number of clusters nc. In consensus weighed clustering, weighted distances are calculated using the cosa2 algorithm with different penalty parameters Lambda. The hyper-parameters are calibrated by maximisation of the consensus score.

Usage

```
Clustering(
  xdata,
  nc = NULL,
  eps = NULL,
  Lambda = NULL,
  K = 100,
  tau = 0.5,
  seed = 1,
  n_cat = 3,
  implementation = HierarchicalClustering,
  scale = TRUE,
  linkage = "complete",
  row = TRUE,
  n_cores = 1,
  output_data = FALSE,
  verbose = TRUE,
  beep = NULL,
  ...
)
```

Arguments

- `xdata` data matrix with observations as rows and variables as columns.
- `nc` matrix of parameters controlling the number of clusters in the underlying algorithm specified in `implementation`. If `nc` is not provided, it is set to `seq(1, tau*nrow(xdata))`.
- `eps` radius in density-based clustering, see `dbscan`. Only used if `implementation=DBSCANClustering`.
- `Lambda` vector of penalty parameters. Only used if `implementation=HierarchicalClustering` or`implementation=PAMClustering`.
- `K` number of resampling iterations.
- `tau` subsample size.
- `seed` value of the seed to initialise the random number generator and ensure reproducibility of the results (see `set.seed`).
- `n_cat` number of categories used to compute the stability score. Possible values are 2 or 3.
- `implementation` function to use for clustering. Possible functions include `HierarchicalClustering` (hierarchical clustering), `PAMClustering` (Partitioning Around Medoids), `KMeansClustering` (k-means) and `GMMClustering` (Gaussian Mixture Models). Alternatively, a user-defined function taking `xdata` and `Lambda` as arguments and returning a binary and symmetric matrix for which diagonal elements are equal to zero can be used.
- `scale` logical indicating if the data should be scaled to ensure that all variables contribute equally to the clustering of the observations.
- `linkage` character string indicating the type of linkage used in hierarchical clustering to define the stable clusters. Possible values include `"complete"`, `"single"` and `"average"` (see argument `"method"` in `hclust` for a full list). Only used if `implementation=HierarchicalClustering`.
- `row` logical indicating if rows (if `row=TRUE`) or columns (if `row=FALSE`) contain the items to cluster.
- `n_cores` number of cores to use for parallel computing (see `mclapply`). Only available on Unix systems.
- `output_data` logical indicating if the input datasets `xdata` and `ydata` should be included in the output.
- `verbose` logical indicating if a loading bar and messages should be printed.
- `beep` sound indicating the end of the run. Possible values are: `NULL` (no sound) or an integer between 1 and 11 (see argument `sound` in `beep`).
- `...` additional parameters passed to the functions provided in `implementation` or `resampling`.

Details

In consensus clustering, a clustering algorithm is applied on `K` subsamples of the observations with different numbers of clusters provided in `nc`. If `row=TRUE` (the default), the observations (rows) are the items to cluster. If `row=FALSE`, the variables (columns) are the items to cluster. For a given number of clusters, the consensus matrix coprop stores the proportion of iterations where two items were in the same estimated cluster, out of all iterations where both items were drawn in the subsample.

Stable cluster membership is obtained by applying a distance-based clustering method using (1-coprop) as distance (see Clusters).

The number of (stable) clusters is calibrated by maximisation of the consensus score (see ConsensusScore) calculated from the observed, most stable, and most unstable likelihoods:

$S(\lambda, n_C) = [log(L_u(\lambda, n_C)) - log(L_o(\lambda, n_C))]/[log(L_u(\lambda, n_C)) - log(L_s(\lambda, n_C))]$

It is strongly recommended to examine the calibration plot (see CalibrationPlot) to check that there is a clear maximum. The absence of a clear maximum suggests that the clustering is not stable, consensus clustering outputs should not be trusted in that case.

To ensure reproducibility of the results, the starting number of the random number generator is set to seed.

For parallelisation, stability selection with different sets of parameters can be run on n_cores cores. This relies on forking with mclapply (specific to Unix systems).

Value

An object of class clustering. A list with:

Sc	a matrix of the best stability scores for different (sets of) parameters controlling the number of clusters and penalisation of attribute weights.
nc	a matrix of numbers of clusters.
Lambda	a matrix of regularisation parameters for attribute weights.
Q	a matrix of the average number of selected attributes by the underlying algorithm with different regularisation parameters.
coprop	an array of consensus matrices. Rows and columns correspond to items. Indices along the third dimension correspond to different parameters controlling the number of clusters and penalisation of attribute weights.
selprop	an array of selection proportions. Columns correspond to attributes. Rows correspond to different parameters controlling the number of clusters and penalisation of attribute weights.
method	a list with type="clustering" and values used for arguments implementation, linkage, and resampling.
params	a list with values used for arguments K, tau, pk, n (number of observations in xdata), and seed.

The rows of Sc, nc, Lambda, Q, selprop and indices along the third dimension of coprop are ordered in the same way and correspond to parameter values stored in nc and Lambda.

References

Bodinier B, Filippi S, Nost TH, Chiquet J, Chadeau-Hyam M (2021). "Automated calibration for stability selection in penalised regression and graphical models: a multi-OMICs network application exploring the molecular response to tobacco smoking." https://arxiv.org/abs/2106.02521.

Kampert MM, Meulman JJ, Friedman JH (2017). "rCOSA: A Software Package for Clustering Objects on Subsets of Attributes." *Journal of Classification*, **34**(3), 514–547. doi: 10.1007/s00357-0179240z.

Friedman JH, Meulman JJ (2004). "Clustering objects on subsets of attributes (with discussion)." *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, **66**(4), 815-849. doi: 10.1111/j.14679868.2004.02059.x, https://rss.onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-9868.2004.02059.x, `https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-9868.2004.02059.x`.

Monti S, Tamayo P, Mesirov J, Golub T (2003). "Consensus Clustering: A Resampling-Based Method for Class Discovery and Visualization of Gene Expression Microarray Data." *Machine Learning*, **52**(1), 91–118. doi: 10.1023/A:1023949509487.

See Also

`Resample`, `StabilityScore`, `HierarchicalClustering`, `PAMClustering`, `KMeansClustering`, `GMMClustering`

Other stability functions: `BiSelection()`, `GraphicalModel()`, `VariableSelection()`

Examples

```
# Consensus clustering
set.seed(1)
simul <- SimulateClustering(
  n = c(30, 30, 30), nu_xc = 1, ev_xc = 0.5
)
stab <- Clustering(xdata = simul$data)
print(stab)
CalibrationPlot(stab)
summary(stab)
Clusters(stab)
plot(stab)

# Consensus weighted clustering
if (requireNamespace("rCOSA", quietly = TRUE)) {
  set.seed(1)
  simul <- SimulateClustering(
    n = c(30, 30, 30), pk = 20,
    theta_xc = c(rep(1, 10), rep(0, 10)),
    ev_xc = 0.9
  )
  stab <- Clustering(
    xdata = simul$data,
    Lambda = LambdaSequence(lmin = 0.1, lmax = 10, cardinal = 10),
    noit = 20, niter = 10
  )
  print(stab)
  CalibrationPlot(stab)
  summary(stab)
  Clusters(stab)
  plot(stab)
  WeightBoxplot(stab)
```

```
}
```

ClusteringAlgo *(Weighted) clustering algorithm*

Description

Runs the (weighted) clustering algorithm specified in the argument `implementation` and returns matrices of variable weights, and the co-membership structure. This function is not using stability.

Usage

```
ClusteringAlgo(
  xdata,
  nc = NULL,
  eps = NULL,
  Lambda = NULL,
  scale = TRUE,
  row = TRUE,
  implementation = HierarchicalClustering,
  ...
)
```

Arguments

`xdata`	data matrix with observations as rows and variables as columns.
`nc`	matrix of parameters controlling the number of clusters in the underlying algorithm specified in `implementation`. If `nc` is not provided, it is set to `seq(1, nrow(xdata))`.
`eps`	radius in density-based clustering, see `dbscan`. Only used if `implementation=DBSCANClustering`.
`Lambda`	vector of penalty parameters.
`scale`	logical indicating if the data should be scaled to ensure that all variables contribute equally to the clustering of the observations.
`row`	logical indicating if rows (if `row=TRUE`) or columns (if `row=FALSE`) contain the items to cluster.
`implementation`	function to use for clustering. Possible functions include `HierarchicalClustering` (hierarchical clustering), `PAMClustering` (Partitioning Around Medoids), `KMeansClustering` (k-means) and `GMMClustering` (Gaussian Mixture Models). Alternatively, a user-defined function taking `xdata` and `Lambda` as arguments and returning a binary and symmetric matrix for which diagonal elements are equal to zero can be used.
`...`	additional parameters passed to the function provided in `implementation`.

Value

A list with:

- `selected` — matrix of binary selection status. Rows correspond to different model parameters. Columns correspond to predictors.
- `weight` — array of model coefficients. Rows correspond to different model parameters. Columns correspond to predictors. Indices along the third dimension correspond to outcome variable(s).
- `comembership` — array of model coefficients. Rows correspond to different model parameters. Columns correspond to predictors. Indices along the third dimension correspond to outcome variable(s).

See Also

VariableSelection

Other underlying algorithm functions: PenalisedGraphical(), PenalisedRegression()

Examples

```
# Simulation of 15 observations belonging to 3 groups
set.seed(1)
simul <- SimulateClustering(
  n = c(5, 5, 5), pk = 100
)

# Running hierarchical clustering
myclust <- ClusteringAlgo(
  xdata = simul$data, nc = 2:5,
  implementation = HierarchicalClustering
)
```

ClusteringPerformance *Clustering performance*

Description

Computes different metrics of clustering performance by comparing true and predicted co-membership. This function can only be used in simulation studies (i.e. when the true cluster membership is known).

Usage

```
ClusteringPerformance(theta, theta_star, ...)
```

Arguments

theta	output from Clustering. Alternatively, it can be the estimated co-membership matrix (see CoMembership).
theta_star	output from SimulateClustering.Alternatively, it can be the true co-membership matrix (see CoMembership).
...	additional arguments to be passed to Clusters.

Value

A matrix of selection metrics including:

TP	number of True Positives (TP)
FN	number of False Negatives (TN)
FP	number of False Positives (FP)
TN	number of True Negatives (TN)
sensitivity	sensitivity, i.e. TP/(TP+FN)
specificity	specificity, i.e. TN/(TN+FP)
accuracy	accuracy, i.e. (TP+TN)/(TP+TN+FP+FN)
precision	precision (p), i.e. TP/(TP+FP)
recall	recall (r), i.e. TP/(TP+FN)
F1_score	F1-score, i.e. 2*p*r/(p+r)
rand	Rand Index, i.e. (TP+TN)/(TP+FP+TN+FN)
ari	Adjusted Rand Index (ARI), i.e. 2*(TP*TN-FP*FN)/((TP+FP)*(TN+FP)+(TP+FN)*(TN+FN))
jaccard	Jaccard index, i.e. TP/(TP+FP+FN)

See Also

Other functions for model performance: SelectionPerformanceGraph(), SelectionPerformance()

Examples

```
# Data simulation
set.seed(1)
simul <- SimulateClustering(
  n = c(30, 30, 30), nu_xc = 1
)
plot(simul)

# Consensus clustering
stab <- Clustering(
  xdata = simul$data, nc = 1:5
)

# Clustering performance
ClusteringPerformance(stab, simul)
```

```
# Alternative formulation
ClusteringPerformance(
  theta = CoMembership(Clusters(stab)),
  theta_star = simul$theta
)
```

Combine *Merging stability selection outputs*

Description

Merges the outputs from two runs of VariableSelection, GraphicalModel or Clustering. The two runs must have been done using the same methods and the same params but with different seeds. The combined output will contain results based on iterations from both stability1 and stability2. This function can be used for parallelisation.

Usage

```
Combine(stability1, stability2, include_beta = TRUE)
```

Arguments

stability1	output from a first run of VariableSelection, GraphicalModel, or Clustering.
stability2	output from a second run of VariableSelection, GraphicalModel, or Clustering.
include_beta	logical indicating if the beta coefficients of visited models should be concatenated. Only applicable to variable selection or clustering.

Value

A single output of the same format.

See Also

VariableSelection, GraphicalModel

Examples

```
## Variable selection

# Data simulation
set.seed(1)
simul <- SimulateRegression(n = 100, pk = 50, family = "gaussian")

# Two runs
```

```
stab1 <- VariableSelection(xdata = simul$xdata, ydata = simul$ydata, seed = 1, K = 10)
stab2 <- VariableSelection(xdata = simul$xdata, ydata = simul$ydata, seed = 2, K = 10)

# Merging the outputs
stab <- Combine(stability1 = stab1, stability2 = stab2, include_beta = FALSE)
str(stab)


## Graphical modelling

# Data simulation
simul <- SimulateGraphical(pk = 20)

# Two runs
stab1 <- GraphicalModel(xdata = simul$data, seed = 1, K = 10)
stab2 <- GraphicalModel(xdata = simul$data, seed = 2, K = 10)

# Merging the outputs
stab <- Combine(stability1 = stab1, stability2 = stab2)
str(stab)


## Clustering

# Data simulation
simul <- SimulateClustering(n = c(15, 15, 15))

# Two runs
stab1 <- Clustering(xdata = simul$data, seed = 1)
stab2 <- Clustering(xdata = simul$data, seed = 2)

# Merging the outputs
stab <- Combine(stability1 = stab1, stability2 = stab2)
str(stab)
```

`CoMembership` *Pairwise co-membership*

Description

Generates a symmetric and binary matrix indicating, if two items are co-members, i.e. belong to the same cluster.

Usage

```
CoMembership(groups)
```

Arguments

`groups`	vector of group membership.

Value

A symmetric and binary matrix.

Examples

```
## Not run:
# Simulated grouping structure
mygroups <- c(rep(1, 3), rep(2, 5), rep(3, 2))

# Co-membership matrix
CoMembership(mygroups)

## End(Not run)
```

`ConsensusScore` *Consensus score*

Description

Computes the consensus score from the consensus matrix. The score measures how unlikely it is that the clustering procedure is uniform (i.e. uninformative) for a given combination of parameters.

Usage

```
ConsensusScore(coprop, nc, K = 100, linkage = "complete")
```

Arguments

`coprop`	consensus matrix obtained with `nc` clusters across `K` subsampling iterations.
`nc`	number of clusters.
`K`	number of subsampling iterations.
`linkage`	character string indicating the type of linkage used in hierarchical clustering to define the stable clusters. Possible values include `"complete"`, `"single"` and `"average"` (see argument `"method"` in `hclust` for a full list).

Details

Let $\Gamma(\lambda, G)$ be the consensus matrix. We introduce the matrix $H(\lambda, G)$ of co-membership count corrected for the subsampling procedure, defined as the integer part of $K\Gamma(\lambda, G)$.

Under the hypothesis of equiprobability of co-membership (instability), we assume that the co-membership counts follow the same binomial distribution for all pairs of items.

Given stable clusters Z and hyper-parameters (λ, G), clustering stability is measured as the probability $p_{\lambda,G}(H|z)$ of observing co-membership counts in H that are at least as high within clusters and at least as low between clusters under equiprobability:

$p_{\lambda,G}(H|Z) = \prod_{i<j} F(H_{ij})^{1_{Z_i =/= Z_j}} \times (1 - F(H_{ij}))^{1_{Z_i = Z_j}}$

where $F(x)$ is the cumulative probability function of the binomial distribution with parameters K and probability $\gamma = N_c/N$ with N_c the number of stable co-members and N the number of item pairs.

This probability is minimised at H^s, which corresponding to the most stable clustering and is defined as:

$H^s_{ij} = K \times 1_{Z_i = Z_j}$

The consensus score is calculated as the following standardised probability:

$S_c(\lambda, G) = (p_{\lambda,G}(H(\lambda, G)|Z))/(p_{\lambda,G}(H^s|Z))$

The consensus score increases with clustering stability.

Value

A consensus score between 0 and 1.

See Also

Other stability metric functions: FDP(), PFER(), StabilityMetrics(), StabilityScore()

Examples

```
# Data simulation
set.seed(2)
simul <- SimulateClustering(
  n = c(30, 30, 30),
  nu_xc = 1
)
plot(simul)

# Consensus clustering
stab <- Clustering(
  xdata = simul$data
)
stab$Sc[3]

# Calculating the consensus score
ConsensusScore(
  coprop = stab$coprop[, , 3],
  nc = stab$nc[3]
)
```

`DBSCANClustering` *(Weighted) density-based clustering*

Description

Runs Density-Based Spatial Clustering of Applications with Noise (DBSCAN) clustering using implementation from `dbscan`. This is also known as the k-medoids algorithm. If `Lambda` is provided, clustering is applied on the weighted distance matrix calculated using the COSA algorithm as implemented in `cosa2`. Otherwise, distances are calculated using `dist`. This function is not using stability.

Usage

```
DBSCANClustering(
  xdata,
  nc = NULL,
  eps = NULL,
  Lambda = NULL,
  distance = "euclidean",
  scale = TRUE,
  ...
)
```

Arguments

`xdata`	data matrix with observations as rows and variables as columns.
`nc`	matrix of parameters controlling the number of clusters in the underlying algorithm specified in `implementation`. If `nc` is not provided, it is set to `seq(1, tau*nrow(xdata))`.
`eps`	radius in density-based clustering, see `dbscan`.
`Lambda`	vector of penalty parameters (see argument `lambda` in `cosa2`). Unweighted distance matrices are used if `Lambda=NULL`.
`distance`	character string indicating the type of distance to use. If `Lambda=NULL`, possible values include `"euclidean"`, `"maximum"`, `"canberra"`, `"binary"`, and `"minkowski"` (see argument `method` in `dist`). Otherwise, possible values include `"euclidean"` (`pwr=2`) or `"absolute"` (`pwr=1`) (see argument `pwr` in `cosa2`).
`scale`	logical indicating if the data should be scaled to ensure that all variables contribute equally to the clustering of the observations.
`...`	additional parameters passed to `dbscan` (except for `minPts` which is fixed to `2`), `dist`, or `cosa2`. If `weighted=TRUE`, parameters `niter` (default to 1) and `noit` (default to 100) correspond to the number of iterations in `cosa2` to calculate weights and may need to be modified.

Value

A list with:

`comembership`	an array of binary and symmetric co-membership matrices.
`weights`	a matrix of median weights by feature.

References

Friedman JH, Meulman JJ (2004). "Clustering objects on subsets of attributes (with discussion)." *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, **66**(4), 815-849. doi: 10.1111/j.14679868.2004.02059.x, https://rss.onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-9868.2004.02059.x, `https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-9868.2004.02059.x`.

Kampert MM, Meulman JJ, Friedman JH (2017). "rCOSA: A Software Package for Clustering Objects on Subsets of Attributes." *Journal of Classification*, **34**(3), 514–547. doi: 10.1007/s00357-0179240z.

See Also

Other clustering algorithms: `GMMClustering()`, `HierarchicalClustering()`, `KMeansClustering()`, `PAMClustering()`

Examples

```
# Data simulation
set.seed(1)
simul <- SimulateClustering(n = c(10, 10), pk = 50)
plot(simul)

# DBSCAN clustering
myclust <- DBSCANClustering(
  xdata = simul$data,
  eps = seq(0, 2 * sqrt(ncol(simul$data) - 1), by = 0.1)
)

# Weighted PAM clustering (using COSA)
if (requireNamespace("rCOSA", quietly = TRUE)) {
  myclust <- DBSCANClustering(
    xdata = simul$data,
    eps = c(0.25, 0.5, 0.75),
    Lambda = c(0.2, 0.5)
  )
}
```

Ensemble *Ensemble model*

Description

Creates an ensemble predictive model from `VariableSelection` outputs.

Usage

```
Ensemble(stability, xdata, ydata)
```

Arguments

`stability`	output of `VariableSelection`.
`xdata`	matrix of predictors with observations as rows and variables as columns.
`ydata`	optional vector or matrix of outcome(s). If `family` is set to `"binomial"` or `"multinomial"`, `ydata` can be a vector with character/numeric values or a factor.

Value

An object of class `ensemble_model`. A list with:

`intercept`	a vector of refitted intercepts for the `K` calibrated models.
`beta`	a matrix of beta coefficients from the `K` calibrated models.
`models`	a list of `K` models that can be used for prediction. These models are of class `"lm"` if `family="gaussian"` or `"glm"` if `family="binomial"`.
`family`	type of regression, extracted from `stability`. Possible values are `"gaussian"` or `"binomial"`.

See Also

Other ensemble model functions: `EnsemblePredictions()`

Examples

```
# Linear regression
set.seed(1)
simul <- SimulateRegression(n = 100, pk = 50, family = "gaussian")
stab <- VariableSelection(xdata = simul$xdata, ydata = simul$ydata, family = "gaussian")
ensemble <- Ensemble(stability = stab, xdata = simul$xdata, ydata = simul$ydata)

# Logistic regression
set.seed(1)
simul <- SimulateRegression(n = 200, pk = 20, family = "binomial")
stab <- VariableSelection(xdata = simul$xdata, ydata = simul$ydata, family = "binomial")
ensemble <- Ensemble(stability = stab, xdata = simul$xdata, ydata = simul$ydata)
```

`EnsemblePredictions` *Predictions from ensemble model*

Description

Makes predictions using an ensemble model created from `VariableSelection` outputs. For each observation in `xdata`, the predictions are calculated as the average predicted values obtained for that observation over the `K` models fitted in calibrated stability selection.

Usage

```
EnsemblePredictions(ensemble, xdata, ...)
```

Arguments

ensemble	output of Ensemble.
xdata	matrix of predictors with observations as rows and variables as columns.
...	additional parameters passed to predict.

Value

A matrix of predictions computed from the observations in xdata.

See Also

predict.variable_selection

Other ensemble model functions: Ensemble()

Examples

```
# Data simulation
set.seed(1)
simul <- SimulateRegression(n = 1000, pk = 50, family = "gaussian")

# Training/test split
ids <- Split(data = simul$ydata, tau = c(0.8, 0.2))
stab <- VariableSelection(
  xdata = simul$xdata[ids[[1]], ],
  ydata = simul$ydata[ids[[1]], ]
)

# Constructing the ensemble model
ensemble <- Ensemble(
  stability = stab,
  xdata = simul$xdata[ids[[1]], ],
  ydata = simul$ydata[ids[[1]], ]
)

# Making predictions
yhat <- EnsemblePredictions(
  ensemble = ensemble,
  xdata = simul$xdata[ids[[2]], ]
)

# Calculating Q-squared
cor(simul$ydata[ids[[2]], ], yhat)^2
```

ExplanatoryPerformance
Prediction performance in regression

Description

Calculates model performance for linear (measured by Q-squared), logistic (AUC) or Cox (C-statistic) regression. This is done by (i) refitting the model on a training set including a proportion tau of the observations, and (ii) evaluating the performance on the remaining observations (test set). For more reliable results, the procedure can be repeated K times (default K=1).

Usage

```
ExplanatoryPerformance(
  xdata,
  ydata,
  new_xdata = NULL,
  new_ydata = NULL,
  stability = NULL,
  family = NULL,
  implementation = NULL,
  prediction = NULL,
  resampling = "subsampling",
  K = 1,
  tau = 0.8,
  seed = 1,
  n_thr = NULL,
  ij_method = FALSE,
  time = 1000,
  verbose = FALSE,
  ...
)
```

Arguments

xdata	matrix of predictors with observations as rows and variables as columns.
ydata	optional vector or matrix of outcome(s). If family is set to "binomial" or "multinomial", ydata can be a vector with character/numeric values or a factor.
new_xdata	optional test set (predictor data).
new_ydata	optional test set (outcome data).
stability	output of VariableSelection. If stability=NULL (the default), a model including all variables in xdata as predictors is fitted. Argument family must be provided in this case.

family	type of regression model. Possible values include "gaussian" (linear regression), "binomial" (logistic regression), "multinomial" (multinomial regression), and "cox" (survival analysis). If provided, this argument must be consistent with input stability.
implementation	optional function to refit the model. If implementation=NULL and stability is the output of VariableSelection, lm (linear regression), coxph (Cox regression), glm (logistic regression), or multinom (multinomial regression) is used.
prediction	optional function to compute predicted values from the model refitted with implementation.
resampling	resampling approach to create the training set. The default is "subsampling" for sampling without replacement of a proportion tau of the observations. Alternatively, this argument can be a function to use for resampling. This function must use arguments named data and tau and return the IDs of observations to be included in the resampled dataset.
K	number of training-test splits. Only used if new_xdata and new_ydata are not provided.
tau	proportion of observations used in the training set. Only used if new_xdata and new_ydata are not provided.
seed	value of the seed to ensure reproducibility of the results. Only used if new_xdata and new_ydata are not provided.
n_thr	number of thresholds to use to construct the ROC curve. If n_thr=NULL, all predicted probability values are iteratively used as thresholds. For faster computations on large data, less thresholds can be used. Only applicable to logistic regression.
ij_method	logical indicating if the analysis should be done for only one refitting/test split with variance of the concordance index should be computed using the infinitesimal jackknife method as implemented in concordance. If ij_method=FALSE (the default), the concordance indices computed for different refitting/test splits are reported. If ij_method=TRUE, the concordance index and estimated confidence interval at level 0.05 are reported. Only applicable to Cox regression.
time	numeric indicating the time for which the survival probabilities are computed. Only applicable to Cox regression.
verbose	logical indicating if a loading bar and messages should be printed.
...	additional parameters passed to the function provided in resampling.

Details

For a fair evaluation of the prediction performance, the data is split into a training set (including a proportion tau of the observations) and test set (remaining observations). The regression model is fitted on the training set and applied on the test set. Performance metrics are computed in the test set by comparing predicted and observed outcomes.

For logistic regression, a Receiver Operating Characteristic (ROC) analysis is performed: the True and False Positive Rates (TPR and FPR), and Area Under the Curve (AUC) are computed for different thresholds in predicted probabilities.

For Cox regression, the Concordance Index (as implemented in concordance) looking at survival probabilities up to a specific time is computed.

For linear regression, the squared correlation between predicted and observed outcome in the test set (Q-squared) is reported.

Value

A list with:

TPR	True Positive Rate (for logistic regression only).
FPR	False Positive Rate (for logistic regression only).
AUC	Area Under the Curve (for logistic regression only).
concordance	Concordance index (for Cox regression only).
lower	lower bound of the confidence interval at level 0.05 for the concordance index calculated using the infinitesimal jackknife (for Cox regression and with ij_method=TRUE).
upper	upper bound of the confidence interval at level 0.05 for the concordance index calculated using the infinitesimal jackknife (for Cox regression and with ij_method=TRUE).
Beta	matrix of estimated beta coefficients across the K iterations. Coefficients are extracted using the coef function.

See Also

VariableSelection, Refit

Other prediction performance functions: Incremental()

Examples

```
# Data simulation
set.seed(1)
simul <- SimulateRegression(
  n = 1000, pk = 10,
  family = "binomial", ev_xy = 0.8
)

# Data split: selection, training and test set
ids <- Split(
  data = simul$ydata,
  family = "binomial",
  tau = c(0.4, 0.3, 0.3)
)
xselect <- simul$xdata[ids[[1]], ]
yselect <- simul$ydata[ids[[1]], ]
xtrain <- simul$xdata[ids[[2]], ]
ytrain <- simul$ydata[ids[[2]], ]
xtest <- simul$xdata[ids[[3]], ]
ytest <- simul$ydata[ids[[3]], ]

# Stability selection
```

```
stab <- VariableSelection(
  xdata = xselect,
  ydata = yselect,
  family = "binomial"
)

# Performances in test set of model refitted in training set
roc <- ExplanatoryPerformance(
  xdata = xtrain, ydata = ytrain,
  new_xdata = xtest, new_ydata = ytest,
  stability = stab
)
plot(roc)
roc$AUC

# Alternative with multiple training/test splits
roc <- ExplanatoryPerformance(
  xdata = rbind(xtrain, xtest),
  ydata = c(ytrain, ytest),
  stability = stab, K = 100
)
plot(roc)
boxplot(roc$AUC)

# Partial Least Squares Discriminant Analysis
stab <- VariableSelection(
  xdata = xselect,
  ydata = yselect,
  implementation = SparsePLS,
  family = "binomial"
)

# Defining wrapping functions for predictions from PLS-DA
PLSDA <- function(xdata, ydata, family = "binomial") {
  model <- mixOmics::plsda(X = xdata, Y = as.factor(ydata), ncomp = 1)
  return(model)
}
PredictPLSDA <- function(xdata, model) {
  xdata <- xdata[, rownames(model$loadings$X), drop = FALSE]
  predicted <- predict(object = model, newdata = xdata)$predict[, 2, 1]
  return(predicted)
}

# Performances with custom models
roc <- ExplanatoryPerformance(
  xdata = rbind(xtrain, xtest),
  ydata = c(ytrain, ytest),
  stability = stab, K = 100,
  implementation = PLSDA, prediction = PredictPLSDA
)
plot(roc)
```

FDP *False Discovery Proportion*

Description

Computes the False Discovery Proportion (upper-bound) as a ratio of the PFER (upper-bound) over the number of stably selected features. In stability selection, the FDP corresponds to the expected proportion of stably selected features that are not relevant to the outcome (i.e. proportion of False Positives among stably selected features).

Usage

```
FDP(selprop, PFER, pi)
```

Arguments

`selprop`	matrix or vector of selection proportions.
`PFER`	Per Family Error Rate.
`pi`	threshold in selection proportions.

Value

The estimated upper-bound in FDP.

See Also

Other stability metric functions: `ConsensusScore()`, `PFER()`, `StabilityMetrics()`, `StabilityScore()`

Examples

```
# Simulating set of selection proportions
selprop <- round(runif(n = 20), digits = 2)

# Computing the FDP with a threshold of 0.8
fdp <- FDP(PFER = 3, selprop = selprop, pi = 0.8)
```

Folds *Splitting observations into folds*

Description

Generates a list of `n_folds` non-overlapping sets of observation IDs (folds).

Usage

```
Folds(data, family = NULL, n_folds = 5)
```

Arguments

data	vector or matrix of data. In regression, this should be the outcome data.
family	type of regression model. This argument is defined as in glmnet. Possible values include "gaussian" (linear regression), "binomial" (logistic regression), "multinomial" (multinomial regression), and "cox" (survival analysis).
n_folds	number of folds.

Details

For categorical outcomes (i.e. family argument is set to "binomial", "multinomial" or "cox"), the split is done such that the proportion of observations from each of the categories in each of the folds is representative of that of the full sample.

Value

A list of length n_folds with sets of non-overlapping observation IDs.

Examples

```
# Splitting into 5 folds
simul <- SimulateRegression()
ids <- Folds(data = simul$ydata)
lapply(ids, length)

# Balanced folds with respect to a binary variable
simul <- SimulateRegression(family = "binomial")
ids <- Folds(data = simul$ydata, family = "binomial")
lapply(ids, FUN = function(x) {
  table(simul$ydata[x, ])
})
```

GMMClustering *Model-based clustering*

Description

Runs clustering with Gaussian Mixture Models (GMM) using implementation from Mclust. This function is not using stability.

Usage

```
GMMClustering(xdata, nc = NULL, scale = TRUE, ...)
```

Arguments

`xdata`	data matrix with observations as rows and variables as columns.
`nc`	matrix of parameters controlling the number of clusters in the underlying algorithm specified in `implementation`. If `nc` is not provided, it is set to `seq(1, tau*nrow(xdata))`.
`scale`	logical indicating if the data should be scaled to ensure that all variables contribute equally to the clustering of the observations.
`...`	additional parameters passed to `Mclust`.

Value

A list with:

`comembership`	an array of binary and symmetric co-membership matrices.
`weights`	a matrix of median weights by feature.

See Also

Other clustering algorithms: `DBSCANClustering()`, `HierarchicalClustering()`, `KMeansClustering()`, `PAMClustering()`

Examples

```
# Data simulation
set.seed(1)
simul <- SimulateClustering(n = c(10, 10), pk = 50)

# Clustering using Gaussian Mixture Models
mygmm <- GMMClustering(xdata = simul$data, nc = 1:30)
```

Graph *Graph visualisation*

Description

Produces an `igraph` object from an adjacency matrix.

Usage

```
Graph(
  adjacency,
  node_label = NULL,
  node_colour = NULL,
  node_shape = NULL,
  edge_colour = "grey60",
  label_colour = "grey20",
```

```
  mode = "undirected",
  weighted = FALSE,
  satellites = FALSE
)
```

Arguments

adjacency	adjacency matrix or output of GraphicalModel.
node_label	optional vector of node labels. This vector must contain as many entries as there are rows/columns in the adjacency matrix and must be in the same order (the order is used to assign labels to nodes).
node_colour	optional vector of node colours. This vector must contain as many entries as there are rows/columns in the adjacency matrix and must be in the same order (the order is used to assign colours to nodes). Integers, named colours or RGB values can be used.
node_shape	optional vector of node shapes. This vector must contain as many entries as there are rows/columns in the adjacency matrix and must be in the same order (the order is used to assign shapes to nodes). Possible values are "circle", "square", "triangle" or "star".
edge_colour	optional character string for edge colour. Integers, named colours or RGB values can be used.
label_colour	optional character string for label colour. Integers, named colours or RGB values can be used.
mode	character string indicating how the adjacency matrix should be interpreted. Possible values include "undirected" or "directed" (see graph_from_adjacency_matrix).
weighted	indicating if entries of the adjacency matrix should define edge width. If weighted=FALSE, an unweighted igraph object is created, all edges have the same width. If weighted=TRUE, edge width is defined by the corresponding value in the adjacency matrix. If weighted=NULL, nodes are linked by as many edges as indicated in the adjacency matrix (integer values are needed).
satellites	logical indicating if unconnected nodes (satellites) should be included in the igraph object.

Details

All functionalities implemented in igraph can be used on the output. These include cosmetic changes for the visualisation, but also various tools for network analysis (including topological properties and community detection).

The R package visNetwork offers interactive network visualisation tools. An igraph object can easily be converted to a visNetwork object (see example below).

For Cytoscape users, the RCy3 package can be used to open the network in Cytoscape.

Value

An igraph object.

See Also

Adjacency, GraphicalModel, igraph manual, visNetwork manual, Cytoscape

Examples

```
## From adjacency matrix

# Un-weighted
adjacency <- SimulateAdjacency(pk = 20, topology = "scale-free")
plot(Graph(adjacency))

# Weighted
adjacency <- adjacency * runif(prod(dim(adjacency)))
adjacency <- adjacency + t(adjacency)
plot(Graph(adjacency, weighted = TRUE))

# Node colours and shapes
plot(Graph(adjacency, weighted = TRUE, node_shape = "star", node_colour = "red"))


## From stability selection outputs

# Graphical model
set.seed(1)
simul <- SimulateGraphical(pk = 20)
stab <- GraphicalModel(xdata = simul$data)
plot(Graph(stab))

# Sparse PLS
set.seed(1)
simul <- SimulateRegression(n = 50, pk = c(5, 5, 5), family = "gaussian")
x <- simul$xdata
y <- simul$ydata
stab <- BiSelection(
  xdata = simul$xdata, ydata = simul$ydata,
  family = "gaussian", ncomp = 3,
  LambdaX = 1:(ncol(x) - 1),
  implementation = SparsePLS
)
plot(Graph(stab))


## Tools from other packages

# Applying some igraph functionalities
adjacency <- SimulateAdjacency(pk = 20, topology = "scale-free")
mygraph <- Graph(adjacency)
igraph::degree(mygraph)
igraph::betweenness(mygraph)
igraph::shortest_paths(mygraph, from = 1, to = 2)
igraph::walktrap.community(mygraph)
```

```
# Interactive view using visNetwork
if (requireNamespace("visNetwork", quietly = TRUE)) {
  vgraph <- mygraph
  igraph::V(vgraph)$shape <- rep("dot", length(igraph::V(vgraph)))
  v <- visNetwork::visIgraph(vgraph)
  mylayout <- as.matrix(v$x$nodes[, c("x", "y")])
  mylayout[, 2] <- -mylayout[, 2]
  plot(mygraph, layout = mylayout)
}

# Opening in Cytoscape using RCy3
if (requireNamespace("RCy3", quietly = TRUE)) {
  # Make sure that Cytoscape is open before running the following line
  # RCy3::createNetworkFromIgraph(mygraph)
}
```

`GraphComparison` *Edge-wise comparison of two graphs*

Description

Generates an `igraph` object representing the common and graph-specific edges.

Usage

```
GraphComparison(
  graph1,
  graph2,
  col = c("tomato", "forestgreen", "navy"),
  lty = c(2, 3, 1),
  node_colour = NULL,
  show_labels = TRUE,
  ...
)
```

Arguments

`graph1`	first graph. Possible inputs are: adjacency matrix, or `igraph` object, or output of `GraphicalModel`, `VariableSelection`, `BiSelection`, or output of `SimulateGraphical`, `SimulateRegression`.
`graph2`	second graph.
`col`	vector of edge colours. The first entry of the vector defines the colour of edges in `graph1` only, second entry is for edges in `graph2` only and third entry is for common edges.
`lty`	vector of line types for edges. The order is defined as for argument `col`.

node_colour optional vector of node colours. This vector must contain as many entries as there are rows/columns in the adjacency matrix and must be in the same order (the order is used to assign colours to nodes). Integers, named colours or RGB values can be used.

show_labels logical indicating if the node labels should be displayed.

... additional arguments to be passed to Graph.

Value

An igraph object.

See Also

SelectionPerformanceGraph

Examples

```
# Data simulation
set.seed(1)
simul1 <- SimulateGraphical(pk = 30)
set.seed(2)
simul2 <- SimulateGraphical(pk = 30)

# Edge-wise comparison of the two graphs
mygraph <- GraphComparison(
  graph1 = simul1,
  graph2 = simul2
)
plot(mygraph, layout = igraph::layout_with_kk(mygraph))
```

GraphicalAlgo *Graphical model algorithm*

Description

Runs the algorithm specified in the argument implementation and returns the estimated adjacency matrix. This function is not using stability.

Usage

```
GraphicalAlgo(
  xdata,
  pk = NULL,
  Lambda,
  Sequential_template = NULL,
  scale = TRUE,
  implementation = PenalisedGraphical,
  start = "cold",
  ...
)
```

Arguments

xdata
: matrix with observations as rows and variables as columns.

pk
: optional vector encoding the grouping structure. Only used for multi-block stability selection where pk indicates the number of variables in each group. If pk=NULL, single-block stability selection is performed.

Lambda
: matrix of parameters controlling the level of sparsity in the underlying feature selection algorithm specified in implementation. If Lambda=NULL and implementation=PenalisedGraphical, LambdaGridGraphical is used to define a relevant grid. Lambda can be provided as a vector or a matrix with length(pk) columns.

Sequential_template
: logical matrix encoding the type of procedure to use for data with multiple blocks in stability selection graphical modelling. For multi-block estimation, the stability selection model is constructed as the union of block-specific stable edges estimated while the others are weakly penalised (TRUE only for the block currently being calibrated and FALSE for other blocks). Other approaches with joint calibration of the blocks are allowed (all entries are set to TRUE).

scale
: logical indicating if the correlation (scale=TRUE) or covariance (scale=FALSE) matrix should be used as input of glassoFast if implementation=PenalisedGraphical. Otherwise, this argument must be used in the function provided in implementation.

implementation
: function to use for graphical modelling. If implementation=PenalisedGraphical, the algorithm implemented in glassoFast is used for regularised estimation of a conditional independence graph. Alternatively, a user-defined function can be provided.

start
: character string indicating if the algorithm should be initialised at the estimated (inverse) covariance with previous penalty parameters (start="warm") or not (start="cold"). Using start="warm" can speed-up the computations, but could lead to convergence issues (in particular with small Lambda_cardinal). Only used for implementation=PenalisedGraphical (see argument "start" in glassoFast).

...
: additional parameters passed to the function provided in implementation.

Details

The use of the procedure from Equation (4) or (5) is controlled by the argument "Sequential_template".

Value

An array with binary and symmetric adjacency matrices along the third dimension.

See Also

GraphicalModel, PenalisedGraphical

Other wrapping functions: SelectionAlgo()

Examples

```
# Data simulation
set.seed(1)
simul <- SimulateGraphical()

# Running graphical LASSO
myglasso <- GraphicalAlgo(
  xdata = simul$data,
  Lambda = cbind(c(0.1, 0.2))
)
```

GraphicalModel *Stability selection graphical model*

Description

Performs stability selection for graphical models. The underlying graphical model (e.g. graphical LASSO) is run with different combinations of parameters controlling the sparsity (e.g. penalty parameter) and thresholds in selection proportions. These two hyper-parameters are jointly calibrated by maximisation of the stability score.

Usage

```
GraphicalModel(
  xdata,
  pk = NULL,
  Lambda = NULL,
  lambda_other_blocks = 0.1,
  pi_list = seq(0.6, 0.9, by = 0.01),
  K = 100,
  tau = 0.5,
  seed = 1,
  n_cat = 3,
  implementation = PenalisedGraphical,
  start = "warm",
  scale = TRUE,
  resampling = "subsampling",
  cpss = FALSE,
  PFER_method = "MB",
  PFER_thr = Inf,
  FDP_thr = Inf,
  Lambda_cardinal = 50,
  lambda_max = NULL,
  lambda_path_factor = 0.001,
  max_density = 0.5,
  n_cores = 1,
  output_data = FALSE,
```

```
  verbose = TRUE,
  beep = NULL,
  ...
)
```

Arguments

xdata
: data matrix with observations as rows and variables as columns. For multi-block stability selection, the variables in data have to be ordered by group.

pk
: optional vector encoding the grouping structure. Only used for multi-block stability selection where `pk` indicates the number of variables in each group. If `pk=NULL`, single-block stability selection is performed.

Lambda
: matrix of parameters controlling the level of sparsity in the underlying feature selection algorithm specified in `implementation`. If `Lambda=NULL` and `implementation=PenalisedGraphical`, `LambdaGridGraphical` is used to define a relevant grid. `Lambda` can be provided as a vector or a matrix with `length(pk)` columns.

lambda_other_blocks
: optional vector of parameters controlling the level of sparsity in neighbour blocks for the multi-block procedure. To use jointly a specific set of parameters for each block, `lambda_other_blocks` must be set to `NULL` (not recommended). Only used for multi-block stability selection, i.e. if `length(pk)>1`.

pi_list
: vector of thresholds in selection proportions. If `n_cat=3`, these values must be `>0.5` and `<1`. If `n_cat=2`, these values must be `>0` and `<1`.

K
: number of resampling iterations.

tau
: subsample size. Only used if `resampling="subsampling"` and `cpss=FALSE`.

seed
: value of the seed to initialise the random number generator and ensure reproducibility of the results (see `set.seed`).

n_cat
: number of categories used to compute the stability score. Possible values are 2 or 3.

implementation
: function to use for graphical modelling. If `implementation=PenalisedGraphical`, the algorithm implemented in `glassoFast` is used for regularised estimation of a conditional independence graph. Alternatively, a user-defined function can be provided.

start
: character string indicating if the algorithm should be initialised at the estimated (inverse) covariance with previous penalty parameters (`start="warm"`) or not (`start="cold"`). Using `start="warm"` can speed-up the computations, but could lead to convergence issues (in particular with small `Lambda_cardinal`). Only used for `implementation=PenalisedGraphical` (see argument `"start"` in `glassoFast`).

scale
: logical indicating if the correlation (`scale=TRUE`) or covariance (`scale=FALSE`) matrix should be used as input of `glassoFast` if `implementation=PenalisedGraphical`. Otherwise, this argument must be used in the function provided in `implementation`.

resampling
: resampling approach. Possible values are: `"subsampling"` for sampling without replacement of a proportion `tau` of the observations, or `"bootstrap"` for

sampling with replacement generating a resampled dataset with as many observations as in the full sample. Alternatively, this argument can be a function to use for resampling. This function must use arguments named `data` and `tau` and return the IDs of observations to be included in the resampled dataset.

`cpss` logical indicating if complementary pair stability selection should be done. For this, the algorithm is applied on two non-overlapping subsets of half of the observations. A feature is considered as selected if it is selected for both subsamples. With this method, the data is split `K/2` times (`K` models are fitted). Only used if `PFER_method="MB"`.

`PFER_method` method used to compute the upper-bound of the expected number of False Positives (or Per Family Error Rate, PFER). If `PFER_method="MB"`, the method proposed by Meinshausen and Bühlmann (2010) is used. If `PFER_method="SS"`, the method proposed by Shah and Samworth (2013) under the assumption of unimodality is used.

`PFER_thr` threshold in PFER for constrained calibration by error control. If `PFER_thr=Inf` and `FDP_thr=Inf`, unconstrained calibration is used (the default).

`FDP_thr` threshold in the expected proportion of falsely selected features (or False Discovery Proportion) for constrained calibration by error control. If `PFER_thr=Inf` and `FDP_thr=Inf`, unconstrained calibration is used (the default).

`Lambda_cardinal`
number of values in the grid of parameters controlling the level of sparsity in the underlying algorithm. Only used if `Lambda=NULL`.

`lambda_max` optional maximum value for the grid in penalty parameters. If `lambda_max=NULL`, the maximum value is set to the maximum covariance in absolute value. Only used if `implementation=PenalisedGraphical` and `Lambda=NULL`.

`lambda_path_factor`
multiplicative factor used to define the minimum value in the grid.

`max_density` threshold on the density. The grid is defined such that the density of the estimated graph does not exceed max_density.

`n_cores` number of cores to use for parallel computing (see `mclapply`). Only available on Unix systems.

`output_data` logical indicating if the input datasets `xdata` and `ydata` should be included in the output.

`verbose` logical indicating if a loading bar and messages should be printed.

`beep` sound indicating the end of the run. Possible values are: `NULL` (no sound) or an integer between 1 and 11 (see argument `sound` in `beep`).

`...` additional parameters passed to the functions provided in `implementation` or `resampling`.

Details

In stability selection, a feature selection algorithm is fitted on `K` subsamples (or bootstrap samples) of the data with different parameters controlling the sparsity (`Lambda`). For a given (set of) sparsity parameter(s), the proportion out of the `K` models in which each feature is selected is calculated. Features with selection proportions above a threshold pi are considered stably selected. The stability selection model is controlled by the sparsity parameter(s) for the underlying algorithm, and the threshold in selection proportion:

$V_{\lambda,\pi} = \{j : p_\lambda(j) \geq \pi\}$

These parameters can be calibrated by maximisation of a stability score (see StabilityScore) derived from the likelihood under the assumption of uniform (uninformative) selection:

$S_{\lambda,\pi} = -log(L_{\lambda,\pi})$

It is strongly recommended to examine the calibration plot carefully to check that the grids of parameters Lambda and pi_list do not restrict the calibration to a region that would not include the global maximum (see CalibrationPlot). In particular, the grid Lambda may need to be extended when the maximum stability is observed on the left or right edges of the calibration heatmap.

To control the expected number of False Positives (Per Family Error Rate) in the results, a threshold PFER_thr can be specified. The optimisation problem is then constrained to sets of parameters that generate models with an upper-bound in PFER below PFER_thr (see Meinshausen and Bühlmann (2010) and Shah and Samworth (2013)).

Possible resampling procedures include defining (i) K subsamples of a proportion tau of the observations, (ii) K bootstrap samples with the full sample size (obtained with replacement), and (iii) K/2 splits of the data in half for complementary pair stability selection (see arguments resampling and cpss). In complementary pair stability selection, a feature is considered selected at a given resampling iteration if it is selected in the two complementary subsamples.

To ensure reproducibility of the results, the starting number of the random number generator is set to seed.

For parallelisation, stability selection with different sets of parameters can be run on n_cores cores. This relies on forking with mclapply (specific to Unix systems). Alternatively, the function can be run manually with different seeds and all other parameters equal. The results can then be combined using Combine.

The generated network can be converted into igraph object using Graph. The R package visNetwork can be used for interactive network visualisation (see examples in Graph).

Value

An object of class graphical_model. A list with:

S	a matrix of the best stability scores for different (sets of) parameters controlling the level of sparsity in the underlying algorithm.
Lambda	a matrix of parameters controlling the level of sparsity in the underlying algorithm.
Q	a matrix of the average number of selected features by the underlying algorithm with different parameters controlling the level of sparsity.
Q_s	a matrix of the calibrated number of stably selected features with different parameters controlling the level of sparsity.
P	a matrix of calibrated thresholds in selection proportions for different parameters controlling the level of sparsity in the underlying algorithm.
PFER	a matrix of upper-bounds in PFER of calibrated stability selection models with different parameters controlling the level of sparsity.

FDP	a matrix of upper-bounds in FDP of calibrated stability selection models with different parameters controlling the level of sparsity.
S_2d	a matrix of stability scores obtained with different combinations of parameters. Columns correspond to different thresholds in selection proportions.
PFER_2d	a matrix of upper-bounds in FDP obtained with different combinations of parameters. Columns correspond to different thresholds in selection proportions. Only returned if length(pk)=1.
FDP_2d	a matrix of upper-bounds in PFER obtained with different combinations of parameters. Columns correspond to different thresholds in selection proportions. Only returned if length(pk)=1.
selprop	an array of selection proportions. Rows and columns correspond to nodes in the graph. Indices along the third dimension correspond to different parameters controlling the level of sparsity in the underlying algorithm.
sign	a matrix of signs of Pearson's correlations estimated from xdata.
method	a list with type="graphical_model" and values used for arguments implementation, start, resampling, cpss and PFER_method.
params	a list with values used for arguments K, pi_list, tau, n_cat, pk, n (number of observations in xdata), PFER_thr, FDP_thr, seed, lambda_other_blocks, and Sequential_template.

The rows of S, Lambda, Q, Q_s, P, PFER, FDP, S_2d, PFER_2d and FDP_2d, and indices along the third dimension of selprop are ordered in the same way and correspond to parameter values stored in Lambda. For multi-block inference, the columns of S, Lambda, Q, Q_s, P, PFER and FDP, and indices along the third dimension of S_2d correspond to the different blocks.

References

Bodinier B, Filippi S, Nost TH, Chiquet J, Chadeau-Hyam M (2021). "Automated calibration for stability selection in penalised regression and graphical models: a multi-OMICs network application exploring the molecular response to tobacco smoking." https://arxiv.org/abs/2106.02521.

Shah RD, Samworth RJ (2013). "Variable selection with error control: another look at stability selection." *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, **75**(1), 55-80. doi: 10.1111/j.14679868.2011.01034.x.

Meinshausen N, Bühlmann P (2010). "Stability selection." *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, **72**(4), 417-473. doi: 10.1111/j.14679868.2010.00740.x.

Friedman J, Hastie T, Tibshirani R (2008). "Sparse inverse covariance estimation with the graphical lasso." *Biostatistics*, **9**(3), 432–441.

See Also

PenalisedGraphical, GraphicalAlgo, LambdaGridGraphical, Resample, StabilityScore Graph, Adjacency,

Other stability functions: BiSelection(), Clustering(), VariableSelection()

Examples

```
oldpar <- par(no.readonly = TRUE)
par(mar = rep(7, 4))

## Single-block stability selection

# Data simulation
set.seed(1)
simul <- SimulateGraphical(n = 100, pk = 20, nu_within = 0.1)

# Stability selection
stab <- GraphicalModel(xdata = simul$data)
print(stab)

# Calibration heatmap
CalibrationPlot(stab)

# Visualisation of the results
summary(stab)
plot(stab)

# Extraction of adjacency matrix or igraph object
Adjacency(stab)
Graph(stab)


## Multi-block stability selection

# Data simulation
set.seed(1)
simul <- SimulateGraphical(pk = c(10, 10))

# Stability selection
stab <- GraphicalModel(xdata = simul$data, pk = c(10, 10), Lambda_cardinal = 10)
print(stab)

# Calibration heatmap
# par(mfrow = c(1, 3))
CalibrationPlot(stab) # Producing three plots

# Visualisation of the results
summary(stab)
plot(stab)

# Multi-parameter stability selection (not recommended)
Lambda <- matrix(c(0.8, 0.6, 0.3, 0.5, 0.4, 0.3, 0.7, 0.5, 0.1), ncol = 3)
stab <- GraphicalModel(
  xdata = simul$data, pk = c(10, 10),
  Lambda = Lambda, lambda_other_blocks = NULL
)
stab$Lambda
```

```
## Example with user-defined function: shrinkage estimation and selection

# Data simulation
set.seed(1)
simul <- SimulateGraphical(n = 100, pk = 20, nu_within = 0.1)

if (requireNamespace("corpcor", quietly = TRUE)) {
  # Writing user-defined algorithm in a portable function
  ShrinkageSelection <- function(xdata, Lambda, ...) {
    mypcor <- corpcor::pcor.shrink(xdata, verbose = FALSE)
    adjacency <- array(NA, dim = c(nrow(mypcor), ncol(mypcor), nrow(Lambda)))
    for (k in 1:nrow(Lambda)) {
      A <- ifelse(abs(mypcor) >= Lambda[k, 1], yes = 1, no = 0)
      diag(A) <- 0
      adjacency[, , k] <- A
    }
    return(list(adjacency = adjacency))
  }

  # Running the algorithm without stability
  myglasso <- GraphicalAlgo(
    xdata = simul$data,
    Lambda = matrix(c(0.05, 0.1), ncol = 1), implementation = ShrinkageSelection
  )

  # Stability selection using shrinkage estimation and selection
  stab <- GraphicalModel(
    xdata = simul$data, Lambda = matrix(c(0.01, 0.05, 0.1), ncol = 1),
    implementation = ShrinkageSelection
  )
  stable_adjacency <- Adjacency(stab)
}


## Example for the detection of block structure

# Data simulation
set.seed(1)
pk <- sample(1:5, size = 5, replace = TRUE)
simul <- SimulateComponents(
  n = 100, pk = pk,
  v_within = c(0.7, 0.8), v_sign = -1
)

# Data visualisation
Heatmap(
  mat = cor(simul$data),
  col = c("navy", "white", "red"),
  legend_range = c(-1, 1)
)
```

```
par(oldpar)
```

GroupPLS *Group Partial Least Squares*

Description

Runs a group Partial Least Squares model using implementation from sgPLS-package. This function is not using stability.

Usage

```
GroupPLS(
  xdata,
  ydata,
  family = "gaussian",
  group_x,
  group_y = NULL,
  Lambda,
  keepX_previous = NULL,
  keepY = NULL,
  ncomp = 1,
  scale = TRUE,
  ...
)
```

Arguments

xdata	matrix of predictors with observations as rows and variables as columns.
ydata	optional vector or matrix of outcome(s). If family is set to "binomial" or "multinomial", ydata can be a vector with character/numeric values or a factor.
family	type of PLS model. If family="gaussian", a group PLS model as defined in gPLS is run (for continuous outcomes). If family="binomial", a PLS-DA model as defined in gPLSda is run (for categorical outcomes).
group_x	vector encoding the grouping structure among predictors. This argument indicates the number of variables in each group.
group_y	optional vector encoding the grouping structure among outcomes. This argument indicates the number of variables in each group.
Lambda	matrix of parameters controlling the number of selected groups at current component, as defined by ncomp.
keepX_previous	number of selected groups in previous components. Only used if ncomp > 1. The argument keepX in sgPLS is obtained by concatenating keepX_previous and Lambda.

`keepY` number of selected groups of outcome variables. This argument is defined as in sgPLS. Only used if `family="gaussian"`.

`ncomp` number of components.

`scale` logical indicating if the data should be scaled (i.e. transformed so that all variables have a standard deviation of one). Only used if `family="gaussian"`.

`...` additional arguments to be passed to gPLS or gPLSda.

Value

A list with:

`selected` matrix of binary selection status. Rows correspond to different model parameters. Columns correspond to predictors.

`beta_full` array of model coefficients. Rows correspond to different model parameters. Columns correspond to predictors (starting with "X") or outcomes (starting with "Y") variables for different components (denoted by "PC").

References

Liquet B, de Micheaux PL, Hejblum BP, Thiébaut R (2016). "Group and sparse group partial least square approaches applied in genomics context." *Bioinformatics*, **32**(1), 35-42. ISSN 1367-4803, doi: 10.1093/bioinformatics/btv535.

See Also

VariableSelection, BiSelection

Other penalised dimensionality reduction functions: SparseGroupPLS(), SparsePCA(), SparsePLS()

Examples

```
## Group PLS
# Data simulation
set.seed(1)
simul <- SimulateRegression(n = 100, pk = 50, q = 3, family = "gaussian")
x <- simul$xdata
y <- simul$ydata

# Running gPLS with 1 group and sparsity of 0.5
mypls <- GroupPLS(
  xdata = x, ydata = y, Lambda = 1, family = "gaussian",
  group_x = c(10, 15, 25),
)

# Running gPLS with groups on outcomes
mypls <- GroupPLS(
  xdata = x, ydata = y, Lambda = 1, family = "gaussian",
  group_x = c(10, 15, 25),
  group_y = c(2, 1), keepY = 1
)
```

```
## Group PLS-DA
# Data simulation
set.seed(1)
simul <- SimulateRegression(n = 100, pk = 50, family = "binomial")

# Running sgPLS-DA with 1 group and sparsity of 0.9
mypls <- GroupPLS(
  xdata = simul$xdata, ydata = simul$ydata, Lambda = 1, family = "binomial",
  group_x = c(10, 15, 25), test = 0
)
```

HierarchicalClustering

(Weighted) hierarchical clustering

Description

Runs hierarchical clustering using implementation from `hclust`. If `Lambda` is provided, clustering is applied on the weighted distance matrix calculated using the `cosa2` algorithm. Otherwise, distances are calculated using `dist`. This function is not using stability.

Usage

```
HierarchicalClustering(
  xdata,
  nc = NULL,
  Lambda = NULL,
  distance = "euclidean",
  linkage = "complete",
  scale = TRUE,
  row = TRUE,
  ...
)
```

Arguments

`xdata`	data matrix with observations as rows and variables as columns.
`nc`	matrix of parameters controlling the number of clusters in the underlying algorithm specified in `implementation`. If `nc` is not provided, it is set to `seq(1, tau*nrow(xdata))`.
`Lambda`	vector of penalty parameters (see argument `lambda` in `cosa2`). Unweighted distance matrices are used if `Lambda=NULL`.
`distance`	character string indicating the type of distance to use. If `Lambda=NULL`, possible values include `"euclidean"`, `"maximum"`, `"canberra"`, `"binary"`, and `"minkowski"` (see argument `method` in `dist`). Otherwise, possible values include `"euclidean"` (`pwr=2`) or `"absolute"` (`pwr=1`) (see argument `pwr` in `cosa2`).

linkage	character string indicating the type of linkage used in hierarchical clustering to define the stable clusters. Possible values include "complete", "single" and "average" (see argument "method" in hclust for a full list). Only used if implementation=HierarchicalClustering.
scale	logical indicating if the data should be scaled to ensure that all variables contribute equally to the clustering of the observations.
row	logical indicating if rows (if row=TRUE) or columns (if row=FALSE) contain the items to cluster.
...	additional parameters passed to hclust, dist, or cosa2. Parameters niter (default to 1) and noit (default to 100) correspond to the number of iterations in cosa2 to calculate weights and may need to be modified. Argument pwr in cosa2 is ignored, please provide distance instead.

Value

A list with:

comembership	an array of binary and symmetric co-membership matrices.
weights	a matrix of median weights by feature.

References

Friedman JH, Meulman JJ (2004). "Clustering objects on subsets of attributes (with discussion)." *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, **66**(4), 815-849. doi: 10.1111/j.14679868.2004.02059.x, https://rss.onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-9868.2004.02059.x, https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-9868.2004.02059.x.

Kampert MM, Meulman JJ, Friedman JH (2017). "rCOSA: A Software Package for Clustering Objects on Subsets of Attributes." *Journal of Classification*, **34**(3), 514–547. doi: 10.1007/s00357-0179240z.

See Also

Other clustering algorithms: DBSCANClustering(), GMMClustering(), KMeansClustering(), PAMClustering()

Examples

```
# Data simulation
set.seed(1)
simul <- SimulateClustering(n = c(10, 10), pk = 50)

# Hierarchical clustering
myhclust <- HierarchicalClustering(
  xdata = simul$data,
  nc = 1:20
)

# Weighted Hierarchical clustering (using COSA)
```

```
if (requireNamespace("rCOSA", quietly = TRUE)) {
  myhclust <- HierarchicalClustering(
    xdata = simul$data,
    weighted = TRUE,
    nc = 1:20,
    Lambda = c(0.2, 0.5)
  )
}
```

Incremental *Incremental prediction performance in regression*

Description

Computes the prediction performance of regression models where predictors are sequentially added by order of decreasing selection proportion. This function can be used to evaluate the marginal contribution of each of the selected predictors over and above more stable predictors. Performances are evaluated as in ExplanatoryPerformance.

Usage

```
Incremental(
  xdata,
  ydata,
  new_xdata = NULL,
  new_ydata = NULL,
  stability = NULL,
  family = NULL,
  implementation = NULL,
  prediction = NULL,
  resampling = "subsampling",
  n_predictors = NULL,
  K = 100,
  tau = 0.8,
  seed = 1,
  n_thr = NULL,
  ij_method = FALSE,
  time = 1000,
  verbose = TRUE,
  ...
)
```

Arguments

xdata	matrix of predictors with observations as rows and variables as columns.
ydata	optional vector or matrix of outcome(s). If `family` is set to `"binomial"` or `"multinomial"`, `ydata` can be a vector with character/numeric values or a factor.

new_xdata	optional test set (predictor data).
new_ydata	optional test set (outcome data).
stability	output of VariableSelection. If stability=NULL (the default), a model including all variables in xdata as predictors is fitted. Argument family must be provided in this case.
family	type of regression model. Possible values include "gaussian" (linear regression), "binomial" (logistic regression), "multinomial" (multinomial regression), and "cox" (survival analysis). If provided, this argument must be consistent with input stability.
implementation	optional function to refit the model. If implementation=NULL and stability is the output of VariableSelection, lm (linear regression), coxph (Cox regression), glm (logistic regression), or multinom (multinomial regression) is used.
prediction	optional function to compute predicted values from the model refitted with implementation.
resampling	resampling approach to create the training set. The default is "subsampling" for sampling without replacement of a proportion tau of the observations. Alternatively, this argument can be a function to use for resampling. This function must use arguments named data and tau and return the IDs of observations to be included in the resampled dataset.
n_predictors	number of predictors to consider.
K	number of training-test splits. Only used if new_xdata and new_ydata are not provided.
tau	proportion of observations used in the training set. Only used if new_xdata and new_ydata are not provided.
seed	value of the seed to ensure reproducibility of the results. Only used if new_xdata and new_ydata are not provided.
n_thr	number of thresholds to use to construct the ROC curve. If n_thr=NULL, all predicted probability values are iteratively used as thresholds. For faster computations on large data, less thresholds can be used. Only applicable to logistic regression.
ij_method	logical indicating if the analysis should be done for only one refitting/test split with variance of the concordance index should be computed using the infinitesimal jackknife method as implemented in concordance. If ij_method=FALSE (the default), the concordance indices computed for different refitting/test splits are reported. If ij_method=TRUE, the concordance index and estimated confidence interval at level 0.05 are reported. Only applicable to Cox regression.
time	numeric indicating the time for which the survival probabilities are computed. Only applicable to Cox regression.
verbose	logical indicating if a loading bar and messages should be printed.
...	additional parameters passed to the function provided in resampling.

Value

An object of class incremental.

For logistic regression, a list with:

FPR A list with, for each of the models (sequentially added predictors), the False Positive Rates for different thresholds (columns) and different data splits (rows).

TPR A list with, for each of the models (sequentially added predictors), the True Positive Rates for different thresholds (columns) and different data splits (rows).

AUC A list with, for each of the models (sequentially added predictors), a vector of Area Under the Curve (AUC) values obtained with different data splits.

Beta Estimated regression coefficients from visited models.

names Names of the predictors by order of inclusion.

stable Binary vector indicating which predictors are stably selected. Only returned if stability is provided.

For Cox regression, a list with:

concordance If ij_method=FALSE, a list with, for each of the models (sequentially added predictors), a vector of concordance indices obtained with different data splits. If ij_method=TRUE, a vector of concordance indices for each of the models (sequentially added predictors).

lower A vector of the lower bound of the confidence interval at level 0.05 for concordance indices for each of the models (sequentially added predictors). Only returned if ij_method=TRUE.

upper A vector of the upper bound of the confidence interval at level 0.05 for concordance indices for each of the models (sequentially added predictors). Only returned if ij_method=TRUE.

Beta Estimated regression coefficients from visited models.

names Names of the predictors by order of inclusion.

stable Binary vector indicating which predictors are stably selected. Only returned if stability is provided.

For linear regression, a list with:

Q_squared A list with, for each of the models (sequentially added predictors), a vector of Q-squared obtained with different data splits.

Beta Estimated regression coefficients from visited models.

names Names of the predictors by order of inclusion.

stable Binary vector indicating which predictors are stably selected. Only returned if stability is provided.

See Also

VariableSelection, Refit

Other prediction performance functions: ExplanatoryPerformance()

Examples

```
# Data simulation
set.seed(1)
simul <- SimulateRegression(
  n = 1000, pk = 10,
  family = "binomial", ev_xy = 0.8
)

# Data split: selection, training and test set
ids <- Split(
  data = simul$ydata,
  family = "binomial",
  tau = c(0.4, 0.3, 0.3)
)
xselect <- simul$xdata[ids[[1]], ]
yselect <- simul$ydata[ids[[1]], ]
xtrain <- simul$xdata[ids[[2]], ]
ytrain <- simul$ydata[ids[[2]], ]
xtest <- simul$xdata[ids[[3]], ]
ytest <- simul$ydata[ids[[3]], ]

# Stability selection
stab <- VariableSelection(
  xdata = xselect,
  ydata = yselect,
  family = "binomial"
)

# Performances in test set of model refitted in training set
incr <- Incremental(
  xdata = xtrain, ydata = ytrain,
  new_xdata = xtest, new_ydata = ytest,
  stability = stab, n_predictors = 10
)
plot(incr)

# Alternative with multiple training/test splits
incr <- Incremental(
  xdata = rbind(xtrain, xtest),
  ydata = c(ytrain, ytest),
  stability = stab, K = 10, n_predictors = 10
)
plot(incr)
```

`KMeansClustering` *K-means clustering*

Description

Runs k-means clustering using implementation from kmeans. This function is not using stability.

Usage

```
KMeansClustering(xdata, nc = NULL, scale = TRUE, ...)
```

Arguments

xdata	data matrix with observations as rows and variables as columns.
nc	matrix of parameters controlling the number of clusters in the underlying algorithm specified in implementation. If nc is not provided, it is set to seq(1, tau*nrow(xdata)).
scale	logical indicating if the data should be scaled to ensure that all variables contribute equally to the clustering of the observations.
...	additional parameters passed to kmeans.

Value

A list with:

comembership	an array of binary and symmetric co-membership matrices.
weights	a matrix of median weights by feature.

See Also

Other clustering algorithms: DBSCANClustering(), GMMClustering(), HierarchicalClustering(), PAMClustering()

Examples

```
# Data simulation
set.seed(1)
simul <- SimulateClustering(n = c(10, 10), pk = 50)

# k-means clustering
mykmeans <- KMeansClustering(xdata = simul$data, nc = 1:20)
```

LambdaGridGraphical — *Grid of penalty parameters (graphical model)*

Description

Generates a relevant grid of penalty parameter values for penalised graphical models.

Usage

```
LambdaGridGraphical(
  xdata,
  pk = NULL,
  lambda_other_blocks = 0.1,
  K = 100,
  tau = 0.5,
  n_cat = 3,
  implementation = PenalisedGraphical,
  start = "cold",
  scale = TRUE,
  resampling = "subsampling",
  PFER_method = "MB",
  PFER_thr = Inf,
  FDP_thr = Inf,
  Lambda_cardinal = 50,
  lambda_max = NULL,
  lambda_path_factor = 0.001,
  max_density = 0.5,
  ...
)
```

Arguments

`xdata` data matrix with observations as rows and variables as columns. For multi-block stability selection, the variables in data have to be ordered by group.

`pk` optional vector encoding the grouping structure. Only used for multi-block stability selection where `pk` indicates the number of variables in each group. If `pk=NULL`, single-block stability selection is performed.

`lambda_other_blocks` optional vector of parameters controlling the level of sparsity in neighbour blocks for the multi-block procedure. To use jointly a specific set of parameters for each block, `lambda_other_blocks` must be set to `NULL` (not recommended). Only used for multi-block stability selection, i.e. if `length(pk)>1`.

`K` number of resampling iterations.

`tau` subsample size. Only used if `resampling="subsampling"` and `cpss=FALSE`.

`n_cat` number of categories used to compute the stability score. Possible values are 2 or 3.

implementation
: function to use for graphical modelling. If implementation=PenalisedGraphical, the algorithm implemented in glassoFast is used for regularised estimation of a conditional independence graph. Alternatively, a user-defined function can be provided.

start
: character string indicating if the algorithm should be initialised at the estimated (inverse) covariance with previous penalty parameters (start="warm") or not (start="cold"). Using start="warm" can speed-up the computations, but could lead to convergence issues (in particular with small Lambda_cardinal). Only used for implementation=PenalisedGraphical (see argument "start" in glassoFast).

scale
: logical indicating if the correlation (scale=TRUE) or covariance (scale=FALSE) matrix should be used as input of glassoFast if implementation=PenalisedGraphical. Otherwise, this argument must be used in the function provided in implementation.

resampling
: resampling approach. Possible values are: "subsampling" for sampling without replacement of a proportion tau of the observations, or "bootstrap" for sampling with replacement generating a resampled dataset with as many observations as in the full sample. Alternatively, this argument can be a function to use for resampling. This function must use arguments named data and tau and return the IDs of observations to be included in the resampled dataset.

PFER_method
: method used to compute the upper-bound of the expected number of False Positives (or Per Family Error Rate, PFER). If PFER_method="MB", the method proposed by Meinshausen and Bühlmann (2010) is used. If PFER_method="SS", the method proposed by Shah and Samworth (2013) under the assumption of unimodality is used.

PFER_thr
: threshold in PFER for constrained calibration by error control. If PFER_thr=Inf and FDP_thr=Inf, unconstrained calibration is used (the default).

FDP_thr
: threshold in the expected proportion of falsely selected features (or False Discovery Proportion) for constrained calibration by error control. If PFER_thr=Inf and FDP_thr=Inf, unconstrained calibration is used (the default).

Lambda_cardinal
: number of values in the grid of parameters controlling the level of sparsity in the underlying algorithm.

lambda_max
: optional maximum value for the grid in penalty parameters. If lambda_max=NULL, the maximum value is set to the maximum covariance in absolute value. Only used if implementation=PenalisedGraphical.

lambda_path_factor
: multiplicative factor used to define the minimum value in the grid.

max_density
: threshold on the density. The grid is defined such that the density of the estimated graph does not exceed max_density.

...
: additional parameters passed to the functions provided in implementation or resampling.

Value

A matrix of lambda values with length(pk) columns and Lambda_cardinal rows.

See Also

Other lambda grid functions: `LambdaGridRegression()`, `LambdaSequence()`

Examples

```
# Single-block simulation
set.seed(1)
simul <- SimulateGraphical()

# Generating grid of 10 values
Lambda <- LambdaGridGraphical(xdata = simul$data, Lambda_cardinal = 10)

# Ensuring PFER < 5
Lambda <- LambdaGridGraphical(xdata = simul$data, Lambda_cardinal = 10, PFER_thr = 5)

# Multi-block simulation
set.seed(1)
simul <- SimulateGraphical(pk = c(10, 10))

# Multi-block grid
Lambda <- LambdaGridGraphical(xdata = simul$data, pk = c(10, 10), Lambda_cardinal = 10)

# Denser neighbouring blocks
Lambda <- LambdaGridGraphical(
  xdata = simul$data, pk = c(10, 10),
  Lambda_cardinal = 10, lambda_other_blocks = 0
)

# Using different neighbour penalties
Lambda <- LambdaGridGraphical(
  xdata = simul$data, pk = c(10, 10),
  Lambda_cardinal = 10, lambda_other_blocks = c(0.1, 0, 0.1)
)
stab <- GraphicalModel(
  xdata = simul$data, pk = c(10, 10),
  Lambda = Lambda, lambda_other_blocks = c(0.1, 0, 0.1)
)
stab$Lambda

# Visiting from empty to full graphs with max_density=1
Lambda <- LambdaGridGraphical(
  xdata = simul$data, pk = c(10, 10),
  Lambda_cardinal = 10, max_density = 1
)
bigblocks <- BlockMatrix(pk = c(10, 10))
bigblocks_vect <- bigblocks[upper.tri(bigblocks)]
N_blocks <- unname(table(bigblocks_vect))
N_blocks # max number of edges per block
stab <- GraphicalModel(xdata = simul$data, pk = c(10, 10), Lambda = Lambda)
apply(stab$Q, 2, max, na.rm = TRUE) # max average number of edges from underlying algo
```

LambdaGridRegression *Grid of penalty parameters (regression model)*

Description

Generates a relevant grid of penalty parameter values for penalised regression using the implementation in glmnet.

Usage

```
LambdaGridRegression(
  xdata,
  ydata,
  tau = 0.5,
  seed = 1,
  family = "gaussian",
  resampling = "subsampling",
  Lambda_cardinal = 100,
  check_input = TRUE,
  ...
)
```

Arguments

xdata	matrix of predictors with observations as rows and variables as columns.
ydata	optional vector or matrix of outcome(s). If family is set to "binomial" or "multinomial", ydata can be a vector with character/numeric values or a factor.
tau	subsample size. Only used if resampling="subsampling" and cpss=FALSE.
seed	value of the seed to initialise the random number generator and ensure reproducibility of the results (see set.seed).
family	type of regression model. This argument is defined as in glmnet. Possible values include "gaussian" (linear regression), "binomial" (logistic regression), "multinomial" (multinomial regression), and "cox" (survival analysis).
resampling	resampling approach. Possible values are: "subsampling" for sampling without replacement of a proportion tau of the observations, or "bootstrap" for sampling with replacement generating a resampled dataset with as many observations as in the full sample. Alternatively, this argument can be a function to use for resampling. This function must use arguments named data and tau and return the IDs of observations to be included in the resampled dataset.
Lambda_cardinal	number of values in the grid of parameters controlling the level of sparsity in the underlying algorithm.
check_input	logical indicating if input values should be checked (recommended).
...	additional parameters passed to the functions provided in implementation or resampling.

Value

A matrix of lambda values with one column and as many rows as indicated in `Lambda_cardinal`.

See Also

Other lambda grid functions: `LambdaGridGraphical()`, `LambdaSequence()`

Examples

```
# Data simulation
set.seed(1)
simul <- SimulateRegression(n = 100, pk = 50, family = "gaussian") # simulated data

# Lambda grid for linear regression
Lambda <- LambdaGridRegression(
  xdata = simul$xdata, ydata = simul$ydata,
  family = "gaussian", Lambda_cardinal = 20
)

# Grid can be used in VariableSelection()
stab <- VariableSelection(
  xdata = simul$xdata, ydata = simul$ydata,
  family = "gaussian", Lambda = Lambda
)
print(SelectedVariables(stab))
```

`LambdaSequence` *Sequence of penalty parameters*

Description

Generates a sequence of penalty parameters from extreme values and the required number of elements. The sequence is defined on the log-scale.

Usage

```
LambdaSequence(lmax, lmin, cardinal = 100)
```

Arguments

`lmax`	maximum value in the grid.
`lmin`	minimum value in the grid.
`cardinal`	number of values in the grid.

Value

A vector with values between "lmin" and "lmax" and as many values as indicated by "cardinal".

See Also

Other lambda grid functions: LambdaGridGraphical(), LambdaGridRegression()

Examples

```
# Grid from extreme values
mygrid <- LambdaSequence(lmax = 0.7, lmin = 0.001, cardinal = 10)
```

PAMClustering *(Weighted) Partitioning Around Medoids*

Description

Runs Partitioning Around Medoids (PAM) clustering using implementation from pam. This is also known as the k-medoids algorithm. If Lambda is provided, clustering is applied on the weighted distance matrix calculated using the COSA algorithm as implemented in cosa2. Otherwise, distances are calculated using dist. This function is not using stability.

Usage

```
PAMClustering(
  xdata,
  nc = NULL,
  Lambda = NULL,
  distance = "euclidean",
  scale = TRUE,
  ...
)
```

Arguments

xdata	data matrix with observations as rows and variables as columns.
nc	matrix of parameters controlling the number of clusters in the underlying algorithm specified in implementation. If nc is not provided, it is set to seq(1, tau*nrow(xdata)).
Lambda	vector of penalty parameters (see argument lambda in cosa2). Unweighted distance matrices are used if Lambda=NULL.
distance	character string indicating the type of distance to use. If Lambda=NULL, possible values include "euclidean", "maximum", "canberra", "binary", and "minkowski" (see argument method in dist). Otherwise, possible values include "euclidean" (pwr=2) or "absolute" (pwr=1) (see argument pwr in cosa2).
scale	logical indicating if the data should be scaled to ensure that all variables contribute equally to the clustering of the observations.
...	additional parameters passed to pam, dist, or cosa2. If weighted=TRUE, parameters niter (default to 1) and noit (default to 100) correspond to the number of iterations in cosa2 to calculate weights and may need to be modified.

Value

A list with:

comembership — an array of binary and symmetric co-membership matrices.

weights — a matrix of median weights by feature.

References

Friedman JH, Meulman JJ (2004). "Clustering objects on subsets of attributes (with discussion)." *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, **66**(4), 815-849. doi: 10.1111/j.14679868.2004.02059.x, https://rss.onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-9868.2004.02059.x, https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-9868.2004.02059.x.

Kampert MM, Meulman JJ, Friedman JH (2017). "rCOSA: A Software Package for Clustering Objects on Subsets of Attributes." *Journal of Classification*, **34**(3), 514–547. doi: 10.1007/s00357-0179240z.

See Also

Other clustering algorithms: DBSCANClustering(), GMMClustering(), HierarchicalClustering(), KMeansClustering()

Examples

```
# Data simulation
set.seed(1)
simul <- SimulateClustering(n = c(10, 10), pk = 50)

# PAM clustering
myclust <- PAMClustering(
  xdata = simul$data,
  nc = 1:20
)

# Weighted PAM clustering (using COSA)
if (requireNamespace("rCOSA", quietly = TRUE)) {
  myclust <- PAMClustering(
    xdata = simul$data,
    nc = 1:20,
    Lambda = c(0.2, 0.5)
  )
}
```

PenalisedGraphical *Graphical LASSO*

Description

Runs the graphical LASSO algorithm for estimation of a Gaussian Graphical Model (GGM). This function is not using stability.

Usage

```
PenalisedGraphical(
  xdata,
  pk = NULL,
  Lambda,
  Sequential_template = NULL,
  scale = TRUE,
  start = "cold",
  output_omega = FALSE,
  ...
)
```

Arguments

xdata	matrix with observations as rows and variables as columns.
pk	optional vector encoding the grouping structure. Only used for multi-block stability selection where pk indicates the number of variables in each group. If pk=NULL, single-block stability selection is performed.
Lambda	matrix of parameters controlling the level of sparsity.
Sequential_template	logical matrix encoding the type of procedure to use for data with multiple blocks in stability selection graphical modelling. For multi-block estimation, the stability selection model is constructed as the union of block-specific stable edges estimated while the others are weakly penalised (TRUE only for the block currently being calibrated and FALSE for other blocks). Other approaches with joint calibration of the blocks are allowed (all entries are set to TRUE).
scale	logical indicating if the correlation (scale=TRUE) or covariance (scale=FALSE) matrix should be used as input of glassoFast if implementation=PenalisedGraphical. Otherwise, this argument must be used in the function provided in implementation.
start	character string indicating if the algorithm should be initialised at the estimated (inverse) covariance with previous penalty parameters (start="warm") or not (start="cold"). Using start="warm" can speed-up the computations, but could lead to convergence issues (in particular with small Lambda_cardinal). Only used for implementation=PenalisedGraphical (see argument "start" in glassoFast).
output_omega	logical indicating if the estimated precision matrices should be stored and returned.

... additional parameters passed to the function provided in `implementation`.

Details

The use of the procedure from Equation (4) or (5) is controlled by the argument "Sequential_template".

Value

An array with binary and symmetric adjacency matrices along the third dimension.

See Also

GraphicalModel

Other underlying algorithm functions: ClusteringAlgo(), PenalisedRegression()

Examples

```
# Data simulation
set.seed(1)
simul <- SimulateGraphical()

# Running graphical LASSO
myglasso <- PenalisedGraphical(
  xdata = simul$data,
  Lambda = matrix(c(0.1, 0.2), ncol = 1)
)

# Returning estimated precision matrix
myglasso <- PenalisedGraphical(
  xdata = simul$data,
  Lambda = matrix(c(0.1, 0.2), ncol = 1),
  output_omega = TRUE
)
```

PenalisedRegression *Penalised regression*

Description

Runs penalised regression using implementation from glmnet. This function is not using stability.

Usage

```
PenalisedRegression(xdata, ydata, Lambda = NULL, family, ...)
```

Arguments

xdata	matrix of predictors with observations as rows and variables as columns.
ydata	optional vector or matrix of outcome(s). If family is set to "binomial" or "multinomial", ydata can be a vector with character/numeric values or a factor.
Lambda	matrix of parameters controlling the level of sparsity.
family	type of regression model. This argument is defined as in glmnet. Possible values include "gaussian" (linear regression), "binomial" (logistic regression), "multinomial" (multinomial regression), and "cox" (survival analysis).
...	additional parameters passed to glmnet.

Value

A list with:

selected	matrix of binary selection status. Rows correspond to different model parameters. Columns correspond to predictors.
beta_full	array of model coefficients. Rows correspond to different model parameters. Columns correspond to predictors. Indices along the third dimension correspond to outcome variable(s).

See Also

SelectionAlgo, VariableSelection

Other underlying algorithm functions: ClusteringAlgo(), PenalisedGraphical()

Examples

```
# Data simulation
set.seed(1)
simul <- SimulateRegression(pk = 50)

# Running the LASSO
mylasso <- PenalisedRegression(
  xdata = simul$xdata, ydata = simul$ydata,
  Lambda = c(0.1, 0.2), family = "gaussian"
)

# Using glmnet arguments
mylasso <- PenalisedRegression(
  xdata = simul$xdata, ydata = simul$ydata,
  Lambda = c(0.1), family = "gaussian",
  penalty.factor = c(rep(0, 10), rep(1, 40))
)
mylasso$beta_full
```

PFER *Per Family Error Rate*

Description

Computes the Per Family Error Rate upper-bound of a stability selection model using the methods proposed by Meinshausen and Bühlmann (2010) or Shah and Samworth (2013). In stability selection, the PFER corresponds to the expected number of stably selected features that are not relevant to the outcome (i.e. False Positives).

Usage

```
PFER(q, pi, N, K, PFER_method = "MB")
```

Arguments

`q`	average number of features selected by the underlying algorithm.
`pi`	threshold in selection proportions.
`N`	total number of features.
`K`	number of resampling iterations.
`PFER_method`	method used to compute the upper-bound of the expected number of False Positives (or Per Family Error Rate, PFER). If `PFER_method="MB"`, the method proposed by Meinshausen and Bühlmann (2010) is used. If `PFER_method="SS"`, the method proposed by Shah and Samworth (2013) under the assumption of unimodality is used.

Value

The estimated upper-bound in PFER.

References

Meinshausen N, Bühlmann P (2010). "Stability selection." *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, **72**(4), 417-473. doi: 10.1111/j.14679868.2010.00740.x.

Shah RD, Samworth RJ (2013). "Variable selection with error control: another look at stability selection." *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, **75**(1), 55-80. doi: 10.1111/j.14679868.2011.01034.x.

See Also

Other stability metric functions: `ConsensusScore()`, `FDP()`, `StabilityMetrics()`, `StabilityScore()`

Examples

```
# Computing PFER for 10/50 selected features and threshold of 0.8
pfer_mb <- PFER(q = 10, pi = 0.8, N = 50, K = 100, PFER_method = "MB")
pfer_ss <- PFER(q = 10, pi = 0.8, N = 50, K = 100, PFER_method = "SS")
```

plot.clustering *Consensus matrix heatmap*

Description

Creates a heatmap of the (calibrated) consensus matrix. See examples in Clustering.

Usage

```
## S3 method for class 'clustering'
plot(
  x,
  linkage = "complete",
  argmax_id = NULL,
  theta = NULL,
  theta_star = NULL,
  col = c("ivory", "navajowhite", "tomato", "darkred"),
  lines = TRUE,
  col.lines = c("blue"),
  lwd.lines = 2,
  tick = TRUE,
  axes = TRUE,
  col.axis = NULL,
  cex.axis = 1,
  xlas = 2,
  ylas = 2,
  bty = "n",
  ...
)
```

Arguments

x	output of Clustering.
linkage	character string indicating the type of linkage used in hierarchical clustering to define the stable clusters. Possible values include "complete", "single" and "average" (see argument "method" in hclust for a full list).
argmax_id	optional indices of hyper-parameters. If argmax_id=NULL, the calibrated hyper-parameters are used.
theta	optional vector of cluster membership. If provided, the ordering of the items should be the same as in Clusters. This argument is used to re-order the consensus matrix.
theta_star	optional vector of true cluster membership. If provided, the ordering of the items should be the same as in Clusters. This argument is used to define item colours.
col	vector of colours.

`lines`	logical indicating if lines separating the clusters provided in `theta` should be displayed.
`col.lines`	colour of the lines separating the clusters.
`lwd.lines`	width of the lines separating the clusters.
`tick`	logical indicating if axis tickmarks should be displayed.
`axes`	logical indicating if item labels should be displayed.
`col.axis`	optional vector of cluster colours.
`cex.axis`	font size for axes.
`xlas`	orientation of labels on the x-axis, as `las` in `par`.
`ylas`	orientation of labels on the y-axis, as `las` in `par`.
`bty`	character string indicating if the box around the plot should be drawn. Possible values include: `"o"` (default, the box is drawn), or `"n"` (no box).
`...`	additional arguments passed to `Heatmap`.

Value

A heatmap.

`plot.incremental` *Plot of incremental performance*

Description

Represents prediction performances upon sequential inclusion of the predictors in a logistic or Cox regression model as produced by `Incremental`. The median and `quantiles` of the performance metric are reported. See examples in `Incremental`.

Usage

```
## S3 method for class 'incremental'
plot(
  x,
  quantiles = c(0.05, 0.95),
  col = c("red", "grey"),
  col.axis = NULL,
  xgrid = FALSE,
  ygrid = FALSE,
  output_data = FALSE,
  ...
)

IncrementalPlot(
  x,
  quantiles = c(0.05, 0.95),
```

```
  col = c("red", "grey"),
  col.axis = NULL,
  xgrid = FALSE,
  ygrid = FALSE,
  output_data = FALSE,
  ...
)

PlotIncremental(
  x,
  quantiles = c(0.05, 0.95),
  col = c("red", "grey"),
  col.axis = NULL,
  xgrid = FALSE,
  ygrid = FALSE,
  output_data = FALSE,
  ...
)
```

Arguments

`x`	output of `Incremental`.
`quantiles`	quantiles defining the lower and upper bounds.
`col`	vector of colours by stable selection status.
`col.axis`	optional vector of label colours by stable selection status.
`xgrid`	logical indicating if a vertical grid should be drawn.
`ygrid`	logical indicating if a horizontal grid should be drawn.
`output_data`	logical indicating if the median and quantiles should be returned in a matrix.
`...`	additional plotting arguments (see `par`).

Value

A plot.

See Also

`Incremental`

`plot.roc_band` *Receiver Operating Characteristic (ROC) band*

Description

Plots the True Positive Rate (TPR) as a function of the False Positive Rate (FPR) for different thresholds in predicted probabilities. If the results from multiple ROC analyses are provided (e.g. output of `ExplanatoryPerformance` with large `K`), the point-wise median is represented and flanked by a transparent band defined by point-wise `quantiles`. See examples in `ROC` and `ExplanatoryPerformance`.

Usage

```
## S3 method for class 'roc_band'
plot(
  x,
  col_band = NULL,
  alpha = 0.5,
  quantiles = c(0.05, 0.95),
  add = FALSE,
  ...
)
```

Arguments

x	output of ROC or ExplanatoryPerformance.
col_band	colour of the band defined by point-wise quantiles.
alpha	level of opacity for the band.
quantiles	point-wise quantiles of the performances defining the band.
add	logical indicating if the curve should be added to the current plot.
...	additional plotting arguments (see par).

Value

A base plot.

See Also

ROC, ExplanatoryPerformance

---

plot.variable_selection *Plot of selection proportions*

Description

Makes a barplot of selection proportions in decreasing order. See examples in VariableSelection.

Usage

```
## S3 method for class 'variable_selection'
plot(
  x,
  col = c("red", "grey"),
  col.axis = NULL,
  col.thr = "darkred",
  lty.thr = 2,
  n_predictors = NULL,
  ...
)
```

Arguments

x	output of VariableSelection.
col	vector of colours by stable selection status.
col.axis	optional vector of label colours by stable selection status.
col.thr	threshold colour.
lty.thr	threshold line type as lty in par.
n_predictors	number of predictors to display.
...	additional plotting arguments (see par).

Value

A plot.

See Also

VariableSelection

PLS *Partial Least Squares 'a la carte'*

Description

Runs a Partial Least Squares (PLS) model in regression mode using algorithm implemented in pls. This function allows for the construction of components based on different sets of predictor and/or outcome variables. This function is not using stability.

Usage

```
PLS(
  xdata,
  ydata,
  selectedX = NULL,
  selectedY = NULL,
  family = "gaussian",
  ncomp = NULL,
  scale = TRUE
)
```

Arguments

xdata	matrix of predictors with observations as rows and variables as columns.
ydata	optional vector or matrix of outcome(s). If family is set to "binomial" or "multinomial", ydata can be a vector with character/numeric values or a factor.

selectedX — binary matrix of size (ncol(xdata) * ncomp). The binary entries indicate which predictors (in rows) contribute to the definition of each component (in columns). If selectedX=NULL, all predictors are selected for all components.

selectedY — binary matrix of size (ncol(ydata) * ncomp). The binary entries indicate which outcomes (in rows) contribute to the definition of each component (in columns). If selectedY=NULL, all outcomes are selected for all components.

family — type of PLS model. Only family="gaussian" is supported. This corresponds to a PLS model as defined in pls (for continuous outcomes).

ncomp — number of components.

scale — logical indicating if the data should be scaled (i.e. transformed so that all variables have a standard deviation of one).

Details

All matrices are defined as in (Wold et al. 2001). The weight matrix Wmat is the equivalent of loadings$X in pls. The loadings matrix Pmat is the equivalent of mat.c in pls. The score matrices Tmat and Qmat are the equivalent of variates$X and variates$Y in pls.

Value

A list with:

Wmat — matrix of X-weights.

Wstar — matrix of transformed X-weights.

Pmat — matrix of X-loadings.

Cmat — matrix of Y-weights.

Tmat — matrix of X-scores.

Umat — matrix of Y-scores.

Qmat — matrix needed for predictions.

Rmat — matrix needed for predictions.

meansX — vector used for centering of predictors, needed for predictions.

sigmaX — vector used for scaling of predictors, needed for predictions.

meansY — vector used for centering of outcomes, needed for predictions.

sigmaY — vector used for scaling of outcomes, needed for predictions.

methods — a list with family and scale values used for the run.

params — a list with selectedX and selectedY values used for the run.

References

Wold S, Sjöström M, Eriksson L (2001). "PLS-regression: a basic tool of chemometrics." *Chemometrics and Intelligent Laboratory Systems*, **58**(2), 109-130. ISSN 0169-7439, doi: 10.1016/S0169-7439(01)001551, PLS Methods.

See Also

VariableSelection, BiSelection

Examples

```
oldpar <- par(no.readonly = TRUE)

# Data simulation
set.seed(1)
simul <- SimulateRegression(n = 200, pk = 15, q = 3, family = "gaussian")
x <- simul$xdata
y <- simul$ydata

# PLS
mypls <- PLS(xdata = x, ydata = y, ncomp = 3)

# Sparse PLS to identify relevant variables
stab <- BiSelection(
  xdata = x, ydata = y,
  family = "gaussian", ncomp = 3,
  LambdaX = 1:(ncol(x) - 1),
  LambdaY = 1:(ncol(y) - 1),
  implementation = SparsePLS,
  n_cat = 2
)
plot(stab)

# Refitting of PLS model
mypls <- PLS(
  xdata = x, ydata = y,
  selectedX = stab$selectedX,
  selectedY = stab$selectedY
)

# Nonzero entries in weights are the same as in selectedX
par(mfrow = c(2, 2))
Heatmap(stab$selectedX,
  legend = FALSE
)
title("Selected in X")
Heatmap(ifelse(mypls$Wmat != 0, yes = 1, no = 0),
  legend = FALSE
)
title("Nonzero entries in Wmat")
Heatmap(stab$selectedY,
  legend = FALSE
)
title("Selected in Y")
Heatmap(ifelse(mypls$Cmat != 0, yes = 1, no = 0),
  legend = FALSE
)
```

```
title("Nonzero entries in Cmat")

# Multilevel PLS
if (requireNamespace("mixOmics", quietly = TRUE)) {
  # Generating random design
  z <- rep(1:50, each = 4)

  # Extracting the within-variability
  x_within <- mixOmics::withinVariation(X = x, design = cbind(z))

  # Running PLS on within-variability
  mypls <- PLS(xdata = x_within, ydata = y, ncomp = 3)
}

par(oldpar)
```

---

predict.variable_selection
Predict method for stability selection

Description

Computes predicted values from the output of VariableSelection.

Usage

```
## S3 method for class 'variable_selection'
predict(
  object,
  xdata,
  ydata,
  newdata = NULL,
  method = c("ensemble", "refit"),
  ...
)
```

Arguments

object	output of VariableSelection.
xdata	predictor data (training set).
ydata	outcome data (training set).
newdata	optional predictor data (test set).
method	character string indicating if predictions should be obtained from an Ensemble model (if method="ensemble") or a Refitted model (if method="refit").
...	additional arguments passed to predict.

Value

Predicted values.

See Also

Refit, Ensemble, EnsemblePredictions

Examples

```
## Linear regression

# Data simulation
set.seed(1)
simul <- SimulateRegression(n = 500, pk = 50, family = "gaussian")

# Training/test split
ids <- Split(data = simul$ydata, tau = c(0.8, 0.2))

# Stability selection
stab <- VariableSelection(
  xdata = simul$xdata[ids[[1]], ],
  ydata = simul$ydata[ids[[1]], ]
)

# Predictions from post stability selection estimation
yhat <- predict(stab,
  xdata = simul$xdata[ids[[1]], ],
  ydata = simul$ydata[ids[[1]], ],
  newdata = simul$xdata[ids[[2]], ],
  method = "refit"
)
cor(simul$ydata[ids[[2]], ], yhat)^2 # Q-squared

# Predictions from ensemble model
yhat <- predict(stab,
  xdata = simul$xdata[ids[[1]], ],
  ydata = simul$ydata[ids[[1]], ],
  newdata = simul$xdata[ids[[2]], ],
  method = "ensemble"
)
cor(simul$ydata[ids[[2]], ], yhat)^2 # Q-squared


## Logistic regression

# Data simulation
set.seed(1)
simul <- SimulateRegression(n = 500, pk = 20, family = "binomial", ev_xy = 0.9)

# Training/test split
ids <- Split(data = simul$ydata, family = "binomial", tau = c(0.8, 0.2))
```

```
# Stability selection
stab <- VariableSelection(
  xdata = simul$xdata[ids[[1]], ],
  ydata = simul$ydata[ids[[1]], ],
  family = "binomial"
)

# Predictions from post stability selection estimation
yhat <- predict(stab,
  xdata = simul$xdata[ids[[1]], ],
  ydata = simul$ydata[ids[[1]], ],
  newdata = simul$xdata[ids[[2]], ],
  method = "refit", type = "response"
)
plot(ROC(predicted = yhat, observed = simul$ydata[ids[[2]], ]))

# Predictions from ensemble model
yhat <- predict(stab,
  xdata = simul$xdata[ids[[1]], ],
  ydata = simul$ydata[ids[[1]], ],
  newdata = simul$xdata[ids[[2]], ],
  method = "ensemble", type = "response"
)
plot(ROC(predicted = yhat, observed = simul$ydata[ids[[2]], ]),
  add = TRUE,
  col = "blue"
)
```

PredictPLS *Partial Least Squares predictions*

Description

Computes predicted values from a Partial Least Squares (PLS) model in regression mode applied on xdata. This function is using the algorithm implemented in predict.pls.

Usage

```
PredictPLS(xdata, model)
```

Arguments

xdata	matrix of predictors with observations as rows and variables as columns.
model	output of PLS.

Value

An array of predicted values.

See Also

PLS

Examples

```
# Data simulation
set.seed(1)
simul <- SimulateRegression(n = 100, pk = c(5, 5, 5), family = "gaussian")
x <- simul$xdata
y <- simul$ydata

# PLS
mypls <- PLS(xdata = x, ydata = y, ncomp = 3)

# Predicted values
predicted <- PredictPLS(xdata = x, model = mypls)
```

Refit *Regression model refitting*

Description

Refits the regression model with stably selected variables as predictors (without penalisation). Variables in `xdata` not evaluated in the stability selection model will automatically be included as predictors.

Usage

```
Refit(
  xdata,
  ydata,
  stability = NULL,
  family = NULL,
  implementation = NULL,
  verbose = TRUE,
  ...
)

Recalibrate(
  xdata,
  ydata,
  stability = NULL,
  family = NULL,
  implementation = NULL,
  verbose = TRUE,
  ...
)
```

Arguments

`xdata`	matrix of predictors with observations as rows and variables as columns.
`ydata`	optional vector or matrix of outcome(s). If `family` is set to `"binomial"` or `"multinomial"`, `ydata` can be a vector with character/numeric values or a factor.
`stability`	output of `VariableSelection` or `BiSelection`. If `stability=NULL` (the default), a model including all variables in `xdata` as predictors is fitted. Argument `family` must be provided in this case.
`family`	type of regression model. Possible values include `"gaussian"` (linear regression), `"binomial"` (logistic regression), `"multinomial"` (multinomial regression), and `"cox"` (survival analysis). If provided, this argument must be consistent with input `stability`.
`implementation`	optional function to refit the model. If `implementation=NULL` and `stability` is the output of `VariableSelection`, `lm` (linear regression), `coxph` (Cox regression), `glm` (logistic regression), or `multinom` (multinomial regression) is used. The function `PLS` is used for the output of `BiSelection`.
`verbose`	logical indicating if a loading bar and messages should be printed.
`...`	additional arguments to be passed to the function provided in `implementation`.

Value

The output as obtained from:

`lm`	for linear regression (`"gaussian"` family).
`coxph`	for Cox regression (`"cox"` family).
`glm`	for logistic regression (`"binomial"` family).
`multinom`	for multinomial regression (`"multinomial"` family).

See Also

`VariableSelection`

Examples

```
## Linear regression

# Data simulation
set.seed(1)
simul <- SimulateRegression(n = 100, pk = 50, family = "gaussian")

# Data split
ids_train <- Resample(
  data = simul$ydata,
  tau = 0.5, family = "gaussian"
)
xtrain <- simul$xdata[ids_train, , drop = FALSE]
```

```
ytrain <- simul$ydata[ids_train, , drop = FALSE]
xrefit <- simul$xdata[-ids_train, , drop = FALSE]
yrefit <- simul$ydata[-ids_train, , drop = FALSE]

# Stability selection
stab <- VariableSelection(xdata = xtrain, ydata = ytrain, family = "gaussian")
print(SelectedVariables(stab))

# Refitting the model
refitted <- Refit(
  xdata = xrefit, ydata = yrefit,
  stability = stab
)
refitted$coefficients # refitted coefficients
head(refitted$fitted.values) # refitted predicted values

# Fitting the full model (including all possible predictors)
refitted <- Refit(
  xdata = simul$xdata, ydata = simul$ydata,
  family = "gaussian"
)
refitted$coefficients # refitted coefficients


## Logistic regression

# Data simulation
set.seed(1)
simul <- SimulateRegression(n = 200, pk = 20, family = "binomial")

# Data split
ids_train <- Resample(
  data = simul$ydata,
  tau = 0.5, family = "binomial"
)
xtrain <- simul$xdata[ids_train, , drop = FALSE]
ytrain <- simul$ydata[ids_train, , drop = FALSE]
xrefit <- simul$xdata[-ids_train, , drop = FALSE]
yrefit <- simul$ydata[-ids_train, , drop = FALSE]

# Stability selection
stab <- VariableSelection(xdata = xtrain, ydata = ytrain, family = "binomial")

# Refitting the model
refitted <- Refit(
  xdata = xrefit, ydata = yrefit,
  stability = stab
)
refitted$coefficients # refitted coefficients
head(refitted$fitted.values) # refitted predicted probabilities


## Partial Least Squares (multiple components)
```

```
# Data simulation
set.seed(1)
simul <- SimulateRegression(n = 500, pk = 15, q = 3, family = "gaussian")

# Data split
ids_train <- Resample(
  data = simul$ydata,
  tau = 0.5, family = "gaussian"
)
xtrain <- simul$xdata[ids_train, , drop = FALSE]
ytrain <- simul$ydata[ids_train, , drop = FALSE]
xrefit <- simul$xdata[-ids_train, , drop = FALSE]
yrefit <- simul$ydata[-ids_train, , drop = FALSE]

# Stability selection
stab <- BiSelection(
  xdata = xtrain, ydata = ytrain,
  family = "gaussian", ncomp = 3,
  LambdaX = 1:(ncol(xtrain) - 1),
  LambdaY = 1:(ncol(ytrain) - 1),
  implementation = SparsePLS
)
plot(stab)

# Refitting the model
refitted <- Refit(
  xdata = xrefit, ydata = yrefit,
  stability = stab
)
refitted$Wmat # refitted X-weights
refitted$Cmat # refitted Y-weights
```

`Resample` *Resampling observations*

Description

Generates a vector of resampled observation IDs.

Usage

```
Resample(data, family = NULL, tau = 0.5, resampling = "subsampling", ...)
```

Arguments

`data`	vector or matrix of data. In regression, this should be the outcome data.

family	type of regression model. This argument is defined as in glmnet. Possible values include "gaussian" (linear regression), "binomial" (logistic regression), "multinomial" (multinomial regression), and "cox" (survival analysis).
tau	subsample size. Only used if resampling="subsampling" and cpss=FALSE.
resampling	resampling approach. Possible values are: "subsampling" for sampling without replacement of a proportion tau of the observations, or "bootstrap" for sampling with replacement generating a resampled dataset with as many observations as in the full sample. Alternatively, this argument can be a function to use for resampling. This function must use arguments named data and tau and return the IDs of observations to be included in the resampled dataset.
...	additional parameters passed to the function provided in resampling.

Details

With categorical outcomes (i.e. "family" argument is set to "binomial", "multinomial" or "cox"), the resampling is done such that the proportion of observations from each of the categories is representative of that of the full sample.

Value

A vector of resampled IDs.

Examples

```
## Linear regression framework
# Data simulation
simul <- SimulateRegression()

# Subsampling
ids <- Resample(data = simul$ydata, family = "gaussian")
sum(duplicated(ids))

# Bootstrapping
ids <- Resample(data = simul$ydata, family = "gaussian", resampling = "bootstrap")
sum(duplicated(ids))

## Logistic regression framework
# Data simulation
simul <- SimulateRegression(family = "binomial")

# Subsampling
ids <- Resample(data = simul$ydata, family = "binomial")
sum(duplicated(ids))
prop.table(table(simul$ydata))
prop.table(table(simul$ydata[ids]))

# Data simulation for a binary confounder
conf <- ifelse(runif(n = 100) > 0.5, yes = 1, no = 0)

# User-defined resampling function
```

```
BalancedResampling <- function(data, tau, Z, ...) {
  s <- NULL
  for (z in unique(Z)) {
   s <- c(s, sample(which((data == "0") & (Z == z)), size = tau * sum((data == "0") & (Z == z))))
   s <- c(s, sample(which((data == "1") & (Z == z)), size = tau * sum((data == "1") & (Z == z))))
  }
  return(s)
}

# Resampling keeping proportions by Y and Z
ids <- Resample(data = simul$ydata, family = "binomial", resampling = BalancedResampling, Z = conf)
prop.table(table(simul$ydata, conf))
prop.table(table(simul$ydata[ids], conf[ids]))

# User-defined resampling for stability selection
stab <- VariableSelection(
  xdata = simul$xdata, ydata = simul$ydata, family = "binomial",
  resampling = BalancedResampling, Z = conf
)
```

`SelectionAlgo` *Variable selection algorithm*

Description

Runs the variable selection algorithm specified in the argument `implementation`. This function is not using stability.

Usage

```
SelectionAlgo(
  xdata,
  ydata = NULL,
  Lambda,
  group_x = NULL,
  family = NULL,
  implementation = PenalisedRegression,
  ...
)
```

Arguments

`xdata`	matrix of predictors with observations as rows and variables as columns.
`ydata`	optional vector or matrix of outcome(s). If `family` is set to `"binomial"` or `"multinomial"`, `ydata` can be a vector with character/numeric values or a factor.

Lambda	matrix of parameters controlling the level of sparsity in the underlying feature selection algorithm specified in implementation. If Lambda=NULL and implementation=PenalisedRegression, LambdaGridRegression is used to define a relevant grid.
group_x	vector encoding the grouping structure among predictors. This argument indicates the number of variables in each group. Only used for models with group penalisation (e.g. implementation=GroupPLS or implementation=SparseGroupPLS).
family	type of regression model. This argument is defined as in glmnet. Possible values include "gaussian" (linear regression), "binomial" (logistic regression), "multinomial" (multinomial regression), and "cox" (survival analysis).
implementation	function to use for variable selection. Possible functions are: PenalisedRegression, SparsePLS, GroupPLS and SparseGroupPLS. Alternatively, a user-defined function can be provided.
...	additional parameters passed to the function provided in implementation.

Value

A list with:

selected	matrix of binary selection status. Rows correspond to different model parameters. Columns correspond to predictors.
beta_full	array of model coefficients. Rows correspond to different model parameters. Columns correspond to predictors. Indices along the third dimension correspond to outcome variable(s).

See Also

VariableSelection, PenalisedRegression, SparsePCA, SparsePLS, GroupPLS, SparseGroupPLS

Other wrapping functions: GraphicalAlgo()

Examples

```
# Data simulation (univariate outcome)
set.seed(1)
simul <- SimulateRegression(pk = 50)

# Running the LASSO
mylasso <- SelectionAlgo(
  xdata = simul$xdata, ydata = simul$ydata,
  Lambda = c(0.1, 0.2), family = "gaussian",
)

# Data simulation (multivariate outcome)
set.seed(1)
simul <- SimulateRegression(pk = 50, q = 3)

# Running multivariate Gaussian LASSO
mylasso <- SelectionAlgo(
  xdata = simul$xdata, ydata = simul$ydata,
```

```
  Lambda = c(0.1, 0.2), family = "mgaussian"
)
str(mylasso)
```

`SelectionPerformance` *Selection performance*

Description

Computes different metrics of selection performance by comparing the set of selected features to the set of true predictors/edges. This function can only be used in simulation studies (i.e. when the true model is known).

Usage

```
SelectionPerformance(theta, theta_star, pk = NULL, cor = NULL, thr = 0.5)
```

Arguments

`theta`	output from `VariableSelection`, `BiSelection`, or `GraphicalModel`. Alternatively, it can be a binary matrix of selected variables (in variable selection) or a binary adjacency matrix (in graphical modelling)
`theta_star`	output from `SimulateRegression`, `SimulateComponents`, or `SimulateGraphical`. Alternatively, it can be a binary matrix of true predictors (in variable selection) or the true binary adjacency matrix (in graphical modelling).
`pk`	optional vector encoding the grouping structure. Only used for multi-block stability selection where `pk` indicates the number of variables in each group. If `pk=NULL`, single-block stability selection is performed.
`cor`	optional correlation matrix. Only used in graphical modelling.
`thr`	optional threshold in correlation. Only used in graphical modelling and when argument "cor" is not NULL.

Value

A matrix of selection metrics including:

`TP`	number of True Positives (TP)
`FN`	number of False Negatives (TN)
`FP`	number of False Positives (FP)
`TN`	number of True Negatives (TN)
`sensitivity`	sensitivity, i.e. TP/(TP+FN)
`specificity`	specificity, i.e. TN/(TN+FP)
`accuracy`	accuracy, i.e. (TP+TN)/(TP+TN+FP+FN)
`precision`	precision (p), i.e. TP/(TP+FP)

`recall`	recall (r), i.e. TP/(TP+FN)
`F1_score`	F1-score, i.e. 2*p*r/(p+r)

If argument "cor" is provided, the number of False Positives among correlated (FP_c) and uncorrelated (FP_i) pairs, defined as having correlations (provided in "cor") above or below the threshold "thr", are also reported.

Block-specific performances are reported if "pk" is not NULL. In this case, the first row of the matrix corresponds to the overall performances, and subsequent rows correspond to each of the blocks. The order of the blocks is defined as in `BlockStructure`.

See Also

Other functions for model performance: `ClusteringPerformance()`, `SelectionPerformanceGraph()`

Examples

```
# Variable selection model
set.seed(1)
simul <- SimulateRegression(pk = 30, nu_xy = 0.5)
stab <- VariableSelection(xdata = simul$xdata, ydata = simul$ydata)

# Selection performance
SelectionPerformance(theta = stab, theta_star = simul)

# Alternative formulation
SelectionPerformance(
  theta = SelectedVariables(stab),
  theta_star = simul$theta
)
```

`SelectionPerformanceGraph` *Graph representation of selection performance*

Description

Generates an igraph object representing the True Positive, False Positive and False Negative edges by comparing the set of selected edges to the set of true edges. This function can only be used in simulation studies (i.e. when the true model is known).

Usage

```
SelectionPerformanceGraph(
  theta,
  theta_star,
  col = c("tomato", "forestgreen", "navy"),
  lty = c(2, 3, 1),
  node_colour = NULL,
  show_labels = TRUE,
  ...
)
```

Arguments

theta	binary adjacency matrix or output of GraphicalModel, VariableSelection, or BiSelection.
theta_star	true binary adjacency matrix or output of SimulateGraphical or SimulateRegression.
col	vector of edge colours. The first entry of the vector defines the colour of False Positive edges, second entry is for True Negatives and third entry is for True Positives.
lty	vector of line types for edges. The order is defined as for argument col.
node_colour	optional vector of node colours. This vector must contain as many entries as there are rows/columns in the adjacency matrix and must be in the same order (the order is used to assign colours to nodes). Integers, named colours or RGB values can be used.
show_labels	logical indicating if the node labels should be displayed.
...	additional arguments to be passed to Graph.

Value

An igraph object.

See Also

SimulateGraphical, SimulateRegression, GraphicalModel, VariableSelection, BiSelection

Other functions for model performance: ClusteringPerformance(), SelectionPerformance()

Examples

```
# Data simulation
set.seed(1)
simul <- SimulateGraphical(pk = 30)

# Stability selection
stab <- GraphicalModel(xdata = simul$data, K = 10)

# Performance graph
```

```
perfgraph <- SelectionPerformanceGraph(
  theta = stab,
  theta_star = simul
)
plot(perfgraph)
```

SelectionProportions *Selection/co-membership proportions*

Description

Extracts selection proportions (for stability selection) or co-membership proportions (for consensus clustering).

Usage

```
SelectionProportions(stability, argmax_id = NULL)

ConsensusMatrix(stability, argmax_id = NULL)
```

Arguments

stability	output of VariableSelection, GraphicalModel, BiSelection, or Clustering.
argmax_id	optional indices of hyper-parameters. If argmax_id=NULL, the calibrated hyper-parameters are used.

Value

A vector or matrix of proportions.

See Also

VariableSelection, GraphicalModel, BiSelection, Clustering

Examples

```
# Stability selection
set.seed(1)
simul <- SimulateRegression(pk = 50)
stab <- VariableSelection(xdata = simul$xdata, ydata = simul$ydata)
SelectionProportions(stab)

# Consensus clustering
set.seed(1)
simul <- SimulateClustering(
  n = c(30, 30, 30), nu_xc = 1, ev_xc = 0.5
```

```
)
stab <- Clustering(xdata = simul$data)
ConsensusMatrix(stab)
```

SparseGroupPLS *Sparse group Partial Least Squares*

Description

Runs a sparse group Partial Least Squares model using implementation from sgPLS-package. This function is not using stability.

Usage

```
SparseGroupPLS(
  xdata,
  ydata,
  family = "gaussian",
  group_x,
  group_y = NULL,
  Lambda,
  alpha.x,
  alpha.y = NULL,
  keepX_previous = NULL,
  keepY = NULL,
  ncomp = 1,
  scale = TRUE,
  ...
)
```

Arguments

xdata	matrix of predictors with observations as rows and variables as columns.
ydata	optional vector or matrix of outcome(s). If family is set to "binomial" or "multinomial", ydata can be a vector with character/numeric values or a factor.
family	type of PLS model. If family="gaussian", a sparse group PLS model as defined in sgPLS is run (for continuous outcomes). If family="binomial", a PLS-DA model as defined in sgPLSda is run (for categorical outcomes).
group_x	vector encoding the grouping structure among predictors. This argument indicates the number of variables in each group.
group_y	optional vector encoding the grouping structure among outcomes. This argument indicates the number of variables in each group.

Lambda	matrix of parameters controlling the number of selected groups at current component, as defined by ncomp.
alpha.x	vector of parameters controlling the level of sparsity within groups of predictors.
alpha.y	optional vector of parameters controlling the level of sparsity within groups of outcomes. Only used if family="gaussian".
keepX_previous	number of selected groups in previous components. Only used if ncomp > 1. The argument keepX in sgPLS is obtained by concatenating keepX_previous and Lambda.
keepY	number of selected groups of outcome variables. This argument is defined as in sgPLS. Only used if family="gaussian".
ncomp	number of components.
scale	logical indicating if the data should be scaled (i.e. transformed so that all variables have a standard deviation of one). Only used if family="gaussian".
...	additional arguments to be passed to sgPLS or sgPLSda.

Value

A list with:

selected	matrix of binary selection status. Rows correspond to different model parameters. Columns correspond to predictors.
beta_full	array of model coefficients. Rows correspond to different model parameters. Columns correspond to predictors (starting with "X") or outcomes (starting with "Y") variables for different components (denoted by "PC").

References

Liquet B, de Micheaux PL, Hejblum BP, Thiébaut R (2016). "Group and sparse group partial least square approaches applied in genomics context." *Bioinformatics*, **32**(1), 35-42. ISSN 1367-4803, doi: 10.1093/bioinformatics/btv535.

See Also

VariableSelection, BiSelection

Other penalised dimensionality reduction functions: GroupPLS(), SparsePCA(), SparsePLS()

Examples

```
## Sparse group PLS
# Data simulation
set.seed(1)
simul <- SimulateRegression(n = 100, pk = 30, q = 3, family = "gaussian")
x <- simul$xdata
y <- simul$ydata

# Running sgPLS with 1 group and sparsity of 0.5
mypls <- SparseGroupPLS(
  xdata = x, ydata = y, Lambda = 1, family = "gaussian",
```

```
  group_x = c(10, 15, 5), alpha.x = 0.5
)

# Running sgPLS with groups on outcomes
mypls <- SparseGroupPLS(
  xdata = x, ydata = y, Lambda = 1, family = "gaussian",
  group_x = c(10, 15, 5), alpha.x = 0.5,
  group_y = c(2, 1), keepY = 1, alpha.y = 0.9
)

## Sparse group PLS-DA
# Data simulation
set.seed(1)
simul <- SimulateRegression(n = 100, pk = 50, family = "binomial")

# Running sgPLS-DA with 1 group and sparsity of 0.9
mypls <- SparseGroupPLS(
  xdata = simul$xdata, ydata = simul$ydata, Lambda = 1, family = "binomial",
  group_x = c(10, 15, 25), alpha.x = 0.9
)
```

SparsePCA *Sparse Principal Component Analysis*

Description

Runs a sparse Principal Component Analysis model using implementation from spca (if algo="sPCA") or spca (if algo="rSVD"). This function is not using stability.

Usage

```
SparsePCA(
  xdata,
  Lambda,
  ncomp = 1,
  scale = TRUE,
  keepX_previous = NULL,
  algorithm = "sPCA",
  ...
)
```

Arguments

xdata	data matrix with observations as rows and variables as columns.
Lambda	matrix of parameters controlling the number of selected variables at current component, as defined by ncomp.
ncomp	number of components.

scale	logical indicating if the data should be scaled (i.e. transformed so that all variables have a standard deviation of one).
keepX_previous	number of selected predictors in previous components. Only used if ncomp > 1.
algorithm	character string indicating the name of the algorithm to use for sparse PCA. Possible values are: "sPCA" (for the algorithm proposed by Zou, Hastie and Tibshirani and implemented in spca) or "rSVD" (for the regularised SVD approach proposed by Shen and Huang and implemented in spca).
...	additional arguments to be passed to spca (if algorithm="sPCA") or spca (if algorithm="rSVD").

Value

A list with:

selected	matrix of binary selection status. Rows correspond to different model parameters. Columns correspond to predictors.
beta_full	array of model coefficients. Rows correspond to different model parameters. Columns correspond to predictors (starting with "X") or outcomes (starting with "Y") variables for different components (denoted by "PC").

References

Zou H, Hastie T, Tibshirani R (2006). "Sparse Principal Component Analysis." *Journal of Computational and Graphical Statistics*, **15**(2), 265-286. doi: 10.1198/106186006X113430.

Shen H, Huang JZ (2008). "Sparse principal component analysis via regularized low rank matrix approximation." *Journal of Multivariate Analysis*, **99**(6), 1015-1034. ISSN 0047-259X, doi: 10.1016/j.jmva.2007.06.007.

See Also

VariableSelection, BiSelection

Other penalised dimensionality reduction functions: GroupPLS(), SparseGroupPLS(), SparsePLS()

Examples

```
# Data simulation
set.seed(1)
simul <- SimulateRegression(n = 100, pk = 50, family = "gaussian")
x <- simul$xdata

# Sparse PCA (by Zou, Hastie, Tibshirani)
mypca <- SparsePCA(xdata = x, ncomp = 2, Lambda = c(1, 2), keepX_previous = 10, algorithm = "sPCA")

# Sparse PCA (by Shen and Huang)
mypca <- SparsePCA(xdata = x, ncomp = 2, Lambda = c(1, 2), keepX_previous = 10, algorithm = "rSVD")
```

SparsePLS *Sparse Partial Least Squares*

Description

Runs a sparse Partial Least Squares model using implementation from sgPLS-package. This function is not using stability.

Usage

```
SparsePLS(
  xdata,
  ydata,
  Lambda,
  family = "gaussian",
  ncomp = 1,
  scale = TRUE,
  keepX_previous = NULL,
  keepY = NULL,
  ...
)
```

Arguments

xdata	matrix of predictors with observations as rows and variables as columns.
ydata	optional vector or matrix of outcome(s). If family is set to "binomial" or "multinomial", ydata can be a vector with character/numeric values or a factor.
Lambda	matrix of parameters controlling the number of selected predictors at current component, as defined by ncomp.
family	type of PLS model. If family="gaussian", a sparse PLS model as defined in sPLS is run (for continuous outcomes). If family="binomial", a PLS-DA model as defined in sPLSda is run (for categorical outcomes).
ncomp	number of components.
scale	logical indicating if the data should be scaled (i.e. transformed so that all variables have a standard deviation of one). Only used if family="gaussian".
keepX_previous	number of selected predictors in previous components. Only used if ncomp > 1. The argument keepX in sPLS is obtained by concatenating keepX_previous and Lambda.
keepY	number of selected outcome variables. This argument is defined as in sPLS. Only used if family="gaussian".
...	additional arguments to be passed to sPLS or sPLSda.

Value

A list with:

`selected` matrix of binary selection status. Rows correspond to different model parameters. Columns correspond to predictors.

`beta_full` array of model coefficients. Rows correspond to different model parameters. Columns correspond to predictors (starting with "X") or outcomes (starting with "Y") variables for different components (denoted by "PC").

References

KA LC, Rossouw D, Robert-Granié C, Besse P (2008). "A sparse PLS for variable selection when integrating omics data." *Stat Appl Genet Mol Biol*, **7**(1), Article 35. ISSN 1544-6115, doi: 10.2202/15446115.1390.

See Also

`VariableSelection`, `BiSelection`

Other penalised dimensionality reduction functions: `GroupPLS()`, `SparseGroupPLS()`, `SparsePCA()`

Examples

```
## Sparse PLS

# Data simulation
set.seed(1)
simul <- SimulateRegression(n = 100, pk = 20, q = 3, family = "gaussian")
x <- simul$xdata
y <- simul$ydata

# Running sPLS with 2 X-variables and 1 Y-variable
mypls <- SparsePLS(xdata = x, ydata = y, Lambda = 2, family = "gaussian", keepY = 1)


## Sparse PLS-DA

# Data simulation
set.seed(1)
simul <- SimulateRegression(n = 100, pk = 20, family = "binomial")

# Running sPLS-DA with 2 X-variables and 1 Y-variable
mypls <- SparsePLS(xdata = simul$xdata, ydata = simul$ydata, Lambda = 2, family = "binomial")
```

Split — *Splitting observations into non-overlapping sets*

Description

Generates a list of `length(tau)` non-overlapping sets of observation IDs.

Usage

```
Split(data, family = NULL, tau = c(0.5, 0.25, 0.25))
```

Arguments

`data`	vector or matrix of data. In regression, this should be the outcome data.
`family`	type of regression model. This argument is defined as in `glmnet`. Possible values include `"gaussian"` (linear regression), `"binomial"` (logistic regression), `"multinomial"` (multinomial regression), and `"cox"` (survival analysis).
`tau`	vector of the proportion of observations in each of the sets.

Details

With categorical outcomes (i.e. `family` argument is set to `"binomial"`, `"multinomial"` or `"cox"`), the split is done such that the proportion of observations from each of the categories in each of the sets is representative of that of the full sample.

Value

A list of length `length(tau)` with sets of non-overlapping observation IDs.

Examples

```
# Splitting into 3 sets
simul <- SimulateRegression()
ids <- Split(data = simul$ydata)
lapply(ids, length)

# Balanced splits with respect to a binary variable
simul <- SimulateRegression(family = "binomial")
ids <- Split(data = simul$ydata, family = "binomial")
lapply(ids, FUN = function(x) {
  table(simul$ydata[x, ])
})
```

Square *Adjacency from bipartite*

Description

Generates a symmetric adjacency matrix encoding a bipartite graph.

Usage

```
Square(x)
```

Arguments

x	matrix encoding the edges between two types of nodes (rows and columns).

Value

A symmetric adjacency matrix encoding a bipartite graph.

Examples

```
# Simulated links between two sets
set.seed(1)
mat <- matrix(sample(c(0, 1), size = 15, replace = TRUE),
  nrow = 5, ncol = 3
)

# Adjacency matrix of a bipartite graph
Square(mat)
```

StabilityMetrics *Stability selection metrics*

Description

Computes the stability score (see StabilityScore) and upper-bounds of the PFER and FDP from selection proportions of models with a given parameter controlling the sparsity of the underlying algorithm and for different thresholds in selection proportions.

Usage

```
StabilityMetrics(
  selprop,
  pk = NULL,
  pi_list = seq(0.6, 0.9, by = 0.01),
  K = 100,
  n_cat = 3,
  PFER_method = "MB",
  PFER_thr_blocks = Inf,
  FDP_thr_blocks = Inf,
  Sequential_template = NULL,
  graph = TRUE,
  group = NULL
)
```

Arguments

selprop
: array of selection proportions.

pk
: optional vector encoding the grouping structure. Only used for multi-block stability selection where pk indicates the number of variables in each group. If pk=NULL, single-block stability selection is performed.

pi_list
: vector of thresholds in selection proportions. If n_cat=3, these values must be >0.5 and <1. If n_cat=2, these values must be >0 and <1.

K
: number of resampling iterations.

n_cat
: number of categories used to compute the stability score. Possible values are 2 or 3.

PFER_method
: method used to compute the upper-bound of the expected number of False Positives (or Per Family Error Rate, PFER). If PFER_method="MB", the method proposed by Meinshausen and Bühlmann (2010) is used. If PFER_method="SS", the method proposed by Shah and Samworth (2013) under the assumption of unimodality is used.

PFER_thr_blocks
: vector of block-specific thresholds in PFER for constrained calibration by error control. If PFER_thr=Inf and FDP_thr=Inf, unconstrained calibration is used.

FDP_thr_blocks
: vector of block-specific thresholds in the expected proportion of falsely selected features (or False Discovery Proportion, FDP) for constrained calibration by error control. If PFER_thr=Inf and FDP_thr=Inf, unconstrained calibration is used.

Sequential_template
: logical matrix encoding the type of procedure to use for data with multiple blocks in stability selection graphical modelling. For multi-block estimation, the stability selection model is constructed as the union of block-specific stable edges estimated while the others are weakly penalised (TRUE only for the block currently being calibrated and FALSE for other blocks). Other approaches with joint calibration of the blocks are allowed (all entries are set to TRUE).

`graph` logical indicating if stability selection is performed in a regression (if `FALSE`) or graphical (if `TRUE`) framework.

`group` vector encoding the grouping structure among predictors. This argument indicates the number of variables in each group and only needs to be provided for group (but not sparse group) penalisation.

Value

A list with:

`S` a matrix of the best (block-specific) stability scores for different (sets of) penalty parameters. In multi-block stability selection, rows correspond to different sets of penalty parameters, (values are stored in the output "Lambda") and columns correspond to different blocks.

`Lambda` a matrix of (block-specific) penalty parameters. In multi-block stability selection, rows correspond to sets of penalty parameters and columns correspond to different blocks.

`Q` a matrix of average numbers of (block-specific) edges selected by the underlying algorithm for different (sets of) penalty parameters. In multi-block stability selection, rows correspond to different sets of penalty parameters, (values are stored in the output "Lambda") and columns correspond to different blocks.

`Q_s` a matrix of calibrated numbers of (block-specific) stable edges for different (sets of) penalty parameters. In multi-block stability selection, rows correspond to different sets of penalty parameters, (values are stored in the output "Lambda") and columns correspond to different blocks.

`P` a matrix of calibrated (block-specific) thresholds in selection proportions for different (sets of) penalty parameters. In multi-block stability selection, rows correspond to different sets of penalty parameters, (values are stored in the output "Lambda") and columns correspond to different blocks.

`PFER` a matrix of computed (block-specific) upper-bounds in PFER of calibrated graphs for different (sets of) penalty parameters. In multi-block stability selection, rows correspond to different sets of penalty parameters, (values are stored in the output "Lambda") and columns correspond to different blocks.

`FDP` a matrix of computed (block-specific) upper-bounds in FDP of calibrated stability selection models for different (sets of) penalty parameters. In multi-block stability selection, rows correspond to different sets of penalty parameters, (values are stored in the output "Lambda") and columns correspond to different blocks.

`S_2d` an array of (block-specific) stability scores obtained with different combinations of parameters. Rows correspond to different (sets of) penalty parameters and columns correspond to different thresholds in selection proportions. In multi-block stability selection, indices along the third dimension correspond to different blocks.

`PFER_2d` an array of computed upper-bounds of PFER obtained with different combinations of parameters. Rows correspond to different penalty parameters and columns correspond to different thresholds in selection proportions. Not available in multi-block stability selection graphical modelling.

FDP_2d — an array of computed upper-bounds of FDP obtained with different combinations of parameters. Rows correspond to different penalty parameters and columns correspond to different thresholds in selection proportions. Not available in multi-block stability selection graphical modelling.

References

Bodinier B, Filippi S, Nost TH, Chiquet J, Chadeau-Hyam M (2021). "Automated calibration for stability selection in penalised regression and graphical models: a multi-OMICs network application exploring the molecular response to tobacco smoking." `https://arxiv.org/abs/2106.02521`.

Meinshausen N, Bühlmann P (2010). "Stability selection." *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, **72**(4), 417-473. doi: 10.1111/j.14679868.2010.00740.x.

Shah RD, Samworth RJ (2013). "Variable selection with error control: another look at stability selection." *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, **75**(1), 55-80. doi: 10.1111/j.14679868.2011.01034.x.

See Also

Other stability metric functions: `ConsensusScore()`, `FDP()`, `PFER()`, `StabilityScore()`

Examples

```
## Sparse or sparse group penalisation

# Simulating set of selection proportions
set.seed(1)
selprop <- matrix(round(runif(n = 20), digits = 2), nrow = 2)

# Computing stability scores for different thresholds
metrics <- StabilityMetrics(
  selprop = selprop, pi = c(0.6, 0.7, 0.8),
  K = 100, graph = FALSE
)


## Group penalisation

# Simulating set of selection proportions
set.seed(1)
selprop <- matrix(round(runif(n = 6), digits = 2), nrow = 2)
selprop <- cbind(
  selprop[, 1], selprop[, 1],
  selprop[, 2], selprop[, 2],
  matrix(rep(selprop[, 3], each = 6), nrow = 2, byrow = TRUE)
)

# Computing stability scores for different thresholds
metrics <- StabilityMetrics(
  selprop = selprop, pi = c(0.6, 0.7, 0.8),
  K = 100, graph = FALSE, group = c(2, 2, 6)
```

```
)
```

`StabilityScore` *Stability score*

Description

Computes the stability score from selection proportions of models with a given parameter controlling the sparsity and for different thresholds in selection proportions. The score measures how unlikely it is that the selection procedure is uniform (i.e. uninformative) for a given combination of parameters.

Usage

```
StabilityScore(
  selprop,
  pi_list = seq(0.6, 0.9, by = 0.01),
  K,
  n_cat = 3,
  group = NULL
)
```

Arguments

`selprop`	array of selection proportions.
`pi_list`	vector of thresholds in selection proportions. If `n_cat=3`, these values must be `>0.5` and `<1`. If `n_cat=2`, these values must be `>0` and `<1`.
`K`	number of resampling iterations.
`n_cat`	number of categories used to compute the stability score. Possible values are 2 or 3.
`group`	vector encoding the grouping structure among predictors. This argument indicates the number of variables in each group and only needs to be provided for group (but not sparse group) penalisation.

Details

The stability score is derived from the likelihood under the assumption of uniform (uninformative) selection.

We classify the features into three categories: the stably selected ones (that have selection proportions $\geq \pi$), the stably excluded ones (selection proportion $\leq 1-\pi$), and the unstable ones (selection proportions between $1-\pi$ and π).

Under the hypothesis of equiprobability of selection (instability), the likelihood of observing stably selected, stably excluded and unstable features can be expressed as:

$L_{\lambda,\pi} = \prod_{j=1}^{N}[(1-F(K\pi-1))^{1_{H_\lambda(j)\geq K\pi}} \times (F(K\pi-1)-F(K(1-\pi))^{1_{(1-\pi)K<H_\lambda(j)<K\pi}} \times F(K(1-\pi))^{1_{H_\lambda(j)\leq K(1-\pi)}}]$

where $H_\lambda(j)$ is the selection count of feature j and $F(x)$ is the cumulative probability function of the binomial distribution with parameters K and the average proportion of selected features over resampling iterations.

The stability score is computed as the minus log-transformed likelihood under the assumption of uniform selection:

$S_{\lambda,\pi} = -log(L_{\lambda,\pi})$

Alternatively, the stability score can be computed by considering only two sets of features: stably selected (selection proportions $\geq \pi$) or not (selection proportions $< \pi$). This can be done using `n_cat=2`.

Value

A vector of stability scores obtained with the different thresholds in selection proportions.

References

Bodinier B, Filippi S, Nost TH, Chiquet J, Chadeau-Hyam M (2021). "Automated calibration for stability selection in penalised regression and graphical models: a multi-OMICs network application exploring the molecular response to tobacco smoking." https://arxiv.org/abs/2106.02521.

See Also

Other stability metric functions: `ConsensusScore()`, `FDP()`, `PFER()`, `StabilityMetrics()`

Examples

```
# Simulating set of selection proportions
set.seed(1)
selprop <- round(runif(n = 20), digits = 2)

# Computing stability scores for different thresholds
score <- StabilityScore(selprop, pi_list = c(0.6, 0.7, 0.8), K = 100)
```

`Stable` *Stable results*

Description

Extracts stable results for stability selection or consensus clustering.

Usage

```
Stable(stability, argmax_id = NULL, linkage = "complete")

SelectedVariables(stability, argmax_id = NULL)

Adjacency(stability, argmax_id = NULL)

Clusters(stability, linkage = "complete", argmax_id = NULL)
```

Arguments

`stability`	output of `VariableSelection`, `BiSelection`, `GraphicalModel` or `Clustering`.
`argmax_id`	optional indices of hyper-parameters. If `argmax_id=NULL`, the calibrated hyper-parameters are used.
`linkage`	character string indicating the type of linkage used in hierarchical clustering to define the stable clusters. Possible values include `"complete"`, `"single"` and `"average"` (see argument `"method"` in `hclust` for a full list).

Value

A binary vector or matrix encoding the selection status (`1` if selected, `0` otherwise) in stability selection or stable cluster membership in consensus clustering.

See Also

`VariableSelection`, `BiSelection`, `GraphicalModel`, `Clustering`

Examples

```
# Variable selection
set.seed(1)
simul <- SimulateRegression(pk = 20)
stab <- VariableSelection(xdata = simul$xdata, ydata = simul$ydata)
SelectedVariables(stab)
Stable(stab)

# Graphical model
set.seed(1)
simul <- SimulateGraphical(pk = 10)
stab <- GraphicalModel(xdata = simul$data)
Adjacency(stab)
Stable(stab)

# Clustering
set.seed(1)
simul <- SimulateClustering(
  n = c(30, 30, 30),
  nu_xc = 1
)
```

```
stab <- Clustering(xdata = simul$data)
Clusters(stab)
Stable(stab)
```

VariableSelection *Stability selection in regression*

Description

Performs stability selection for regression models. The underlying variable selection algorithm (e.g. LASSO regression) is run with different combinations of parameters controlling the sparsity (e.g. penalty parameter) and thresholds in selection proportions. These two hyper-parameters are jointly calibrated by maximisation of the stability score.

Usage

```
VariableSelection(
  xdata,
  ydata = NULL,
  Lambda = NULL,
  pi_list = seq(0.6, 0.9, by = 0.01),
  K = 100,
  tau = 0.5,
  seed = 1,
  n_cat = 3,
  family = "gaussian",
  implementation = PenalisedRegression,
  resampling = "subsampling",
  cpss = FALSE,
  PFER_method = "MB",
  PFER_thr = Inf,
  FDP_thr = Inf,
  Lambda_cardinal = 100,
  group_x = NULL,
  group_penalisation = FALSE,
  n_cores = 1,
  output_data = FALSE,
  verbose = TRUE,
  beep = NULL,
  ...
)
```

Arguments

xdata	matrix of predictors with observations as rows and variables as columns.

ydata
: optional vector or matrix of outcome(s). If family is set to "binomial" or "multinomial", ydata can be a vector with character/numeric values or a factor.

Lambda
: matrix of parameters controlling the level of sparsity in the underlying feature selection algorithm specified in implementation. If Lambda=NULL and implementation=PenalisedRegression, LambdaGridRegression is used to define a relevant grid.

pi_list
: vector of thresholds in selection proportions. If n_cat=3, these values must be >0.5 and <1. If n_cat=2, these values must be >0 and <1.

K
: number of resampling iterations.

tau
: subsample size. Only used if resampling="subsampling" and cpss=FALSE.

seed
: value of the seed to initialise the random number generator and ensure reproducibility of the results (see set.seed).

n_cat
: number of categories used to compute the stability score. Possible values are 2 or 3.

family
: type of regression model. This argument is defined as in glmnet. Possible values include "gaussian" (linear regression), "binomial" (logistic regression), "multinomial" (multinomial regression), and "cox" (survival analysis).

implementation
: function to use for variable selection. Possible functions are: PenalisedRegression, SparsePLS, GroupPLS and SparseGroupPLS. Alternatively, a user-defined function can be provided.

resampling
: resampling approach. Possible values are: "subsampling" for sampling without replacement of a proportion tau of the observations, or "bootstrap" for sampling with replacement generating a resampled dataset with as many observations as in the full sample. Alternatively, this argument can be a function to use for resampling. This function must use arguments named data and tau and return the IDs of observations to be included in the resampled dataset.

cpss
: logical indicating if complementary pair stability selection should be done. For this, the algorithm is applied on two non-overlapping subsets of half of the observations. A feature is considered as selected if it is selected for both subsamples. With this method, the data is split K/2 times (K models are fitted). Only used if PFER_method="MB".

PFER_method
: method used to compute the upper-bound of the expected number of False Positives (or Per Family Error Rate, PFER). If PFER_method="MB", the method proposed by Meinshausen and Bühlmann (2010) is used. If PFER_method="SS", the method proposed by Shah and Samworth (2013) under the assumption of unimodality is used.

PFER_thr
: threshold in PFER for constrained calibration by error control. If PFER_thr=Inf and FDP_thr=Inf, unconstrained calibration is used (the default).

FDP_thr
: threshold in the expected proportion of falsely selected features (or False Discovery Proportion) for constrained calibration by error control. If PFER_thr=Inf and FDP_thr=Inf, unconstrained calibration is used (the default).

Lambda_cardinal
: number of values in the grid of parameters controlling the level of sparsity in the underlying algorithm. Only used if Lambda=NULL.

group_x vector encoding the grouping structure among predictors. This argument indicates the number of variables in each group. Only used for models with group penalisation (e.g. implementation=GroupPLS or implementation=SparseGroupPLS).

group_penalisation logical indicating if a group penalisation should be considered in the stability score. The use of group_penalisation=TRUE strictly applies to group (not sparse-group) penalisation.

n_cores number of cores to use for parallel computing (see mclapply). Only available on Unix systems.

output_data logical indicating if the input datasets xdata and ydata should be included in the output.

verbose logical indicating if a loading bar and messages should be printed.

beep sound indicating the end of the run. Possible values are: NULL (no sound) or an integer between 1 and 11 (see argument sound in beep).

... additional parameters passed to the functions provided in implementation or resampling.

Details

In stability selection, a feature selection algorithm is fitted on K subsamples (or bootstrap samples) of the data with different parameters controlling the sparsity (Lambda). For a given (set of) sparsity parameter(s), the proportion out of the K models in which each feature is selected is calculated. Features with selection proportions above a threshold pi are considered stably selected. The stability selection model is controlled by the sparsity parameter(s) for the underlying algorithm, and the threshold in selection proportion:

$V_{\lambda,\pi} = \{j : p_\lambda(j) \geq \pi\}$

If argument group_penalisation=FALSE, "feature" refers to variable (variable selection model). If argument group_penalisation=TRUE, "feature" refers to group (group selection model). In this case, groups need to be defined *a priori* and specified in argument group_x.

These parameters can be calibrated by maximisation of a stability score (see StabilityScore) derived from the likelihood under the assumption of uniform (uninformative) selection:

$S_{\lambda,\pi} = -log(L_{\lambda,\pi})$

It is strongly recommended to examine the calibration plot carefully to check that the grids of parameters Lambda and pi_list do not restrict the calibration to a region that would not include the global maximum (see CalibrationPlot). In particular, the grid Lambda may need to be extended when the maximum stability is observed on the left or right edges of the calibration heatmap.

To control the expected number of False Positives (Per Family Error Rate) in the results, a threshold PFER_thr can be specified. The optimisation problem is then constrained to sets of parameters that generate models with an upper-bound in PFER below PFER_thr (see Meinshausen and Bühlmann (2010) and Shah and Samworth (2013)).

Possible resampling procedures include defining (i) K subsamples of a proportion tau of the observations, (ii) K bootstrap samples with the full sample size (obtained with replacement), and (iii) K/2 splits of the data in half for complementary pair stability selection (see arguments resampling and cpss). In complementary pair stability selection, a feature is considered selected at a given resampling iteration if it is selected in the two complementary subsamples.

For categorical or time to event outcomes (argument family is "binomial", "multinomial" or "cox"), the proportions of observations from each category in all subsamples or bootstrap samples are the same as in the full sample.

To ensure reproducibility of the results, the starting number of the random number generator is set to seed.

For parallelisation, stability selection with different sets of parameters can be run on n_cores cores. This relies on forking with mclapply (specific to Unix systems). Alternatively, the function can be run manually with different seeds and all other parameters equal. The results can then be combined using Combine.

Value

An object of class variable_selection. A list with:

S
: a matrix of the best stability scores for different parameters controlling the level of sparsity in the underlying algorithm.

Lambda
: a matrix of parameters controlling the level of sparsity in the underlying algorithm.

Q
: a matrix of the average number of selected features by the underlying algorithm with different parameters controlling the level of sparsity.

Q_s
: a matrix of the calibrated number of stably selected features with different parameters controlling the level of sparsity.

P
: a matrix of calibrated thresholds in selection proportions for different parameters controlling the level of sparsity in the underlying algorithm.

PFER
: a matrix of upper-bounds in PFER of calibrated stability selection models with different parameters controlling the level of sparsity.

FDP
: a matrix of upper-bounds in FDP of calibrated stability selection models with different parameters controlling the level of sparsity.

S_2d
: a matrix of stability scores obtained with different combinations of parameters. Columns correspond to different thresholds in selection proportions.

PFER_2d
: a matrix of upper-bounds in FDP obtained with different combinations of parameters. Columns correspond to different thresholds in selection proportions.

FDP_2d
: a matrix of upper-bounds in PFER obtained with different combinations of parameters. Columns correspond to different thresholds in selection proportions.

selprop
: a matrix of selection proportions. Columns correspond to predictors from xdata.

Beta
: an array of model coefficients. Columns correspond to predictors from xdata. Indices along the third dimension correspond to different resampling iterations. With multivariate outcomes, indices along the fourth dimension correspond to outcome-specific coefficients.

method
: a list with type="variable_selection" and values used for arguments implementation, family, resampling, cpss and PFER_method.

params
: a list with values used for arguments K, pi_list, tau, n_cat, pk, n (number of observations), PFER_thr, FDP_thr and seed. The datasets xdata and ydata are also included if output_data=TRUE.

For all matrices and arrays returned, the rows are ordered in the same way and correspond to parameter values stored in Lambda.

References

Bodinier B, Filippi S, Nost TH, Chiquet J, Chadeau-Hyam M (2021). "Automated calibration for stability selection in penalised regression and graphical models: a multi-OMICs network application exploring the molecular response to tobacco smoking." https://arxiv.org/abs/2106.02521.

Shah RD, Samworth RJ (2013). "Variable selection with error control: another look at stability selection." *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, **75**(1), 55-80. doi: 10.1111/j.14679868.2011.01034.x.

Meinshausen N, Bühlmann P (2010). "Stability selection." *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, **72**(4), 417-473. doi: 10.1111/j.14679868.2010.00740.x.

Tibshirani R (1996). "Regression Shrinkage and Selection via the Lasso." *Journal of the Royal Statistical Society. Series B (Methodological)*, **58**(1), 267–288. ISSN 00359246, http://www.jstor.org/stable/2346178.

See Also

PenalisedRegression, SelectionAlgo, LambdaGridRegression, Resample, StabilityScore Refit, ExplanatoryPerformance, Incremental,

Other stability functions: BiSelection(), Clustering(), GraphicalModel()

Examples

```
oldpar <- par(no.readonly = TRUE)
par(mar = rep(7, 4))

# Linear regression
set.seed(1)
simul <- SimulateRegression(n = 100, pk = 50, family = "gaussian")
stab <- VariableSelection(xdata = simul$xdata, ydata = simul$ydata, family = "gaussian")
print(stab)
CalibrationPlot(stab)
summary(stab)
SelectedVariables(stab)
plot(stab)

# Using additional arguments from glmnet (e.g. penalty.factor)
stab <- VariableSelection(
  xdata = simul$xdata, ydata = simul$ydata, family = "gaussian",
  penalty.factor = c(rep(1, 45), rep(0, 5))
)
summary(stab)

# Regression with multivariate outcomes
set.seed(1)
simul <- SimulateRegression(n = 100, pk = 20, q = 3, family = "gaussian")
stab <- VariableSelection(xdata = simul$xdata, ydata = simul$ydata, family = "mgaussian")
summary(stab)
```

```
# Logistic regression
set.seed(1)
simul <- SimulateRegression(n = 200, pk = 10, family = "binomial", ev_xy = 0.8)
stab <- VariableSelection(xdata = simul$xdata, ydata = simul$ydata, family = "binomial")
summary(stab)

# Sparse PCA (1 component, see BiSelection for more components)
set.seed(1)
simul <- SimulateComponents(pk = c(5, 3, 4))
stab <- VariableSelection(
  xdata = simul$data,
  Lambda = 1:(ncol(simul$data) - 1),
  implementation = SparsePCA
)
CalibrationPlot(stab, xlab = "")
summary(stab)

# Sparse PLS (1 outcome, 1 component, see BiSelection for more options)
set.seed(1)
simul <- SimulateRegression(n = 100, pk = 50, family = "gaussian")
stab <- VariableSelection(
  xdata = simul$xdata, ydata = simul$ydata,
  Lambda = 1:(ncol(simul$xdata) - 1),
  implementation = SparsePLS, family = "gaussian"
)
CalibrationPlot(stab, xlab = "")
SelectedVariables(stab)

# Group PLS (1 outcome, 1 component, see BiSelection for more options)
stab <- VariableSelection(
  xdata = simul$xdata, ydata = simul$ydata,
  Lambda = 1:5,
  group_x = c(5, 5, 10, 20, 10),
  group_penalisation = TRUE,
  implementation = GroupPLS, family = "gaussian"
)
CalibrationPlot(stab, xlab = "")
SelectedVariables(stab)

# Sparse PLS-DA (1 outcome, 1 component, see BiSelection for more options)
set.seed(1)
simul <- SimulateRegression(n = 200, pk = 20, family = "binomial")
stab <- VariableSelection(
  xdata = simul$xdata, ydata = simul$ydata,
  Lambda = 1:(ncol(simul$xdata) - 1),
  implementation = SparsePLS,
  family = "binomial"
)
CalibrationPlot(stab, xlab = "")
summary(stab)

# Example with more hyper-parameters: elastic net
set.seed(1)
```

```
simul <- SimulateRegression(n = 100, pk = 50, family = "gaussian")
TuneElasticNet <- function(xdata, ydata, family, alpha) {
  stab <- VariableSelection(
    xdata = xdata, ydata = ydata,
    family = family, alpha = alpha, verbose = FALSE
  )
  return(max(stab$S, na.rm = TRUE))
}
myopt <- optimise(TuneElasticNet,
  lower = 0.1, upper = 1, maximum = TRUE,
  xdata = simul$xdata, ydata = simul$ydata,
  family = "gaussian"
)
stab <- VariableSelection(
  xdata = simul$xdata, ydata = simul$ydata,
  family = "gaussian", alpha = myopt$maximum
)
summary(stab)
enet <- SelectedVariables(stab)

# Comparison with LASSO
stab <- VariableSelection(xdata = simul$xdata, ydata = simul$ydata, family = "gaussian")
summary(stab)
lasso <- SelectedVariables(stab)
table(lasso, enet)

# Example using an external function: group-LASSO with gglasso
if (requireNamespace("gglasso", quietly = TRUE)) {
  set.seed(1)
  simul <- SimulateRegression(n = 200, pk = 20, family = "binomial")
  ManualGridGroupLasso <- function(xdata, ydata, family, group_x, ...) {
    # Defining the grouping
    group <- do.call(c, lapply(1:length(group_x), FUN = function(i) {
      rep(i, group_x[i])
    }))

    if (family == "binomial") {
      ytmp <- ydata
      ytmp[ytmp == min(ytmp)] <- -1
      ytmp[ytmp == max(ytmp)] <- 1
      return(gglasso::gglasso(xdata, ytmp, loss = "logit", group = group, ...))
    } else {
    return(gglasso::gglasso(xdata, ydata, lambda = lambda, loss = "ls", group = group, ...))
    }
  }
  Lambda <- LambdaGridRegression(
    xdata = simul$xdata, ydata = simul$ydata,
    family = "binomial", Lambda_cardinal = 20,
    implementation = ManualGridGroupLasso,
    group_x = rep(5, 4)
  )
  GroupLasso <- function(xdata, ydata, Lambda, family, group_x, ...) {
    # Defining the grouping
```

```
    group <- do.call(c, lapply(1:length(group_x), FUN = function(i) {
      rep(i, group_x[i])
    }))

    # Running the regression
    if (family == "binomial") {
      ytmp <- ydata
      ytmp[ytmp == min(ytmp)] <- -1
      ytmp[ytmp == max(ytmp)] <- 1
    mymodel <- gglasso::gglasso(xdata, ytmp, lambda = Lambda, loss = "logit", group = group, ...)
    }
    if (family == "gaussian") {
    mymodel <- gglasso::gglasso(xdata, ydata, lambda = Lambda, loss = "ls", group = group, ...)
    }
    # Extracting and formatting the beta coefficients
    beta_full <- t(as.matrix(mymodel$beta))
    beta_full <- beta_full[, colnames(xdata)]

    selected <- ifelse(beta_full != 0, yes = 1, no = 0)

    return(list(selected = selected, beta_full = beta_full))
  }
  stab <- VariableSelection(
    xdata = simul$xdata, ydata = simul$ydata,
    implementation = GroupLasso, family = "binomial", Lambda = Lambda,
    group_x = rep(5, 4),
    group_penalisation = TRUE
  )
  summary(stab)
}

par(oldpar)
```

WeightBoxplot *Stable attribute weights*

Description

Creates a boxplots of the distribution of (calibrated) median attribute weights obtained from the COSA algorithm across the subsampling iterations. See examples in Clustering.

Usage

```
WeightBoxplot(
  stability,
  at = NULL,
  argmax_id = NULL,
  col = NULL,
```

```
  boxwex = 0.3,
  xlab = "",
  ylab = "Weight",
  cex.lab = 1.5,
  las = 3,
  frame = "F",
  add = FALSE,
  ...
)
```

Arguments

stability	output of Clustering.
at	coordinates along the x-axis (more details in boxplot).
argmax_id	optional indices of hyper-parameters. If argmax_id=NULL, the calibrated hyper-parameters are used.
col	optional vector of colours.
boxwex	box width (more details in boxplot).
xlab	label of the x-axis.
ylab	label of the y-axis.
cex.lab	font size for labels.
las	orientation of labels on the x-axis (see par).
frame	logical indicating if the box around the plot should be drawn (more details in boxplot).
add	logical indicating if the boxplot should be added to the current plot.
...	additional parameters passed to boxplot).

Value

A boxplot.

See Also

Clustering

Index