

# Package ‘sharp’

October 14, 2022

**Type** Package

**Title** Stability-enHanced Approaches using Resampling Procedures

**Version** 1.2.0

**Date** 2022-08-12

**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** Implementation of stability selection for graphical modelling and variable selection in regression and dimensionality reduction. These models use on resampling approaches to estimate selection probabilities (N Meinshausen, P Bühlmann (2010) <[doi:10.1111/j.1467-9868.2010.00740.x](https://doi.org/10.1111/j.1467-9868.2010.00740.x)>). Calibration of the hyper-parameters is done via maximisation of a stability score measuring the likelihood of informative (non-uniform) selection (B Bodinier, S Filippi, TH Nost, J Chiquet, M Chadeau-Hyam (2021) <[arXiv:2106.02521](https://arxiv.org/abs/2106.02521)>).

**License** GPL (>= 3)

**Language** en-GB

**Encoding** UTF-8

**RoxygenNote** 7.2.0

**Depends** fake

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

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

**Config/testthat/edition** 3

**RdMacros** Rdpack

**NeedsCompilation** no

**Repository** CRAN

**Date/Publication** 2022-08-15 07:40:02 UTC

**R topics documented:**

|                                     |    |
|-------------------------------------|----|
| sharp-package . . . . .             | 3  |
| Adjacency . . . . .                 | 5  |
| AggregatedEffects . . . . .         | 6  |
| Argmax . . . . .                    | 7  |
| ArgmaxId . . . . .                  | 8  |
| BiSelection . . . . .               | 9  |
| BlockLambdaGrid . . . . .           | 17 |
| CalibrationPlot . . . . .           | 18 |
| Combine . . . . .                   | 22 |
| Ensemble . . . . .                  | 23 |
| EnsemblePredictions . . . . .       | 24 |
| ExplanatoryPerformance . . . . .    | 25 |
| FDP . . . . .                       | 31 |
| Folds . . . . .                     | 31 |
| Graph . . . . .                     | 32 |
| GraphComparison . . . . .           | 35 |
| GraphicalAlgo . . . . .             | 36 |
| GraphicalModel . . . . .            | 38 |
| GroupPLS . . . . .                  | 44 |
| Incremental . . . . .               | 47 |
| LambdaGridGraphical . . . . .       | 51 |
| LambdaGridRegression . . . . .      | 54 |
| LambdaSequence . . . . .            | 56 |
| PenalisedGraphical . . . . .        | 57 |
| PenalisedRegression . . . . .       | 58 |
| PFER . . . . .                      | 60 |
| PlotIncremental . . . . .           | 61 |
| PlotROC . . . . .                   | 64 |
| PLS . . . . .                       | 66 |
| PredictPLS . . . . .                | 69 |
| Refit . . . . .                     | 70 |
| Resample . . . . .                  | 75 |
| ROC . . . . .                       | 76 |
| SelectedVariables . . . . .         | 78 |
| SelectionAlgo . . . . .             | 79 |
| SelectionPerformance . . . . .      | 80 |
| SelectionPerformanceGraph . . . . . | 83 |
| SelectionProportions . . . . .      | 85 |
| SparseGroupPLS . . . . .            | 87 |
| SparsePCA . . . . .                 | 89 |
| SparsePLS . . . . .                 | 91 |
| Split . . . . .                     | 93 |
| Square . . . . .                    | 94 |
| StabilityMetrics . . . . .          | 94 |
| StabilityScore . . . . .            | 98 |
| VariableSelection . . . . .         | 99 |

---

|               |   |
|---------------|---|
| sharp-package | <i>sharp: Stability-enhanced Approaches using Resampling Procedures</i> |
|---------------|---|

---

## Description

Implementation of stability selection for graphical modelling and variable selection in regression and dimensionality reduction. These models use on resampling approaches to estimate selection probabilities. Calibration of the hyper-parameters is done via maximisation of a stability score measuring the likelihood of informative (non-uniform) selection.

## Details

|             |  |
|-------------|--|
| Package:    | sharp  |
| Type:       | Package                                      |
| Version:    | 0.1  |
| Date:       | 2021-04-30                                   |
| 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>.

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](https://doi.org/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](https://doi.org/10.1111/j.14679868.2010.00740.x).

## 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)
```

---

Adjacency

*Calibrated adjacency matrix*

---

### Description

Extracts the adjacency matrix of the (calibrated) stability selection graphical model.

### Usage

```
Adjacency(stability, argmax_id = NULL)
```

### Arguments

|                        |  |
|------------------------|--|
| <code>stability</code> | output of <a href="#">GraphicalModel</a> .   |
| <code>argmax_id</code> | optional matrix of parameter IDs. If <code>argmax_id=NULL</code> , the calibrated model is used. |

### Value

A binary and symmetric adjacency matrix encoding an undirected graph with no self-loops.

### See Also

[GraphicalModel](#)

Other calibration functions: [ArgmaxId\(\)](#), [Argmax\(\)](#), [CalibrationPlot\(\)](#), [SelectedVariables\(\)](#), [SelectionProportions\(\)](#)

### Examples

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

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

# Calibrated adjacency matrix
A <- Adjacency(stab)

# User-defined parameters
myids <- matrix(c(20, 10), nrow = 1)
stab$Lambda[myids[1], 1] # corresponding penalty
stab$params$pi_list[myids[2]] # corresponding threshold
A <- Adjacency(stab, argmax_id = myids)
```

---

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 <a href="#">VariableSelection</a> or <a href="#">BiSelection</a> .   |
| 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 <a href="#">median</a> or <a href="#">mean</a> . |
| ...       | 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 = c(20, 30), 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 = c(5, 5, 5), 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)

```

---

Argmax

*Calibrated parameters*


---

## Description

Extracts calibrated parameter values in stability selection.

## Usage

```
Argmax(stability)
```

**Arguments**

stability      output of [VariableSelection](#), [BiSelection](#) or [GraphicalModel](#).

**Value**

A matrix of parameter values. If applied to the output of [VariableSelection](#) or [GraphicalModel](#), the first column (lambda) denotes the calibrated hyper-parameter of the underlying algorithm. The second column (pi) is the calibrated threshold in selection/co-membership proportions. For multi-block graphical models, rows correspond to different blocks. If applied to the output of [BiSelection](#), all columns are named as in object summary.

**See Also**

[VariableSelection](#), [GraphicalModel](#), [BiSelection](#)

Other calibration functions: [Adjacency\(\)](#), [ArgmaxId\(\)](#), [CalibrationPlot\(\)](#), [SelectedVariables\(\)](#), [SelectionProportions\(\)](#)

**Examples**

```
## Graphical modelling

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

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

# Extracting calibrated parameters
Argmax(stab)
```

---

ArgmaxId

*Calibrated parameter indices*

---

**Description**

Extracts the indices of calibrated parameters with respect to the grids provided in `Lambda` and `pi_list` in `stability`.

**Usage**

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

**Arguments**

- `stability` output of [VariableSelection](#) or [GraphicalModel](#). If `stability=NULL`, `S` must be provided.
- `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 parameter indices. For multi-block graphical models, rows correspond to different blocks.

**See Also**

[VariableSelection](#), [GraphicalModel](#)

Other calibration functions: [Adjacency\(\)](#), [Argmax\(\)](#), [CalibrationPlot\(\)](#), [SelectedVariables\(\)](#), [SelectionProportions\(\)](#)

**Examples**

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

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

# Extracting IDs of calibrated parameters
ids <- ArgmaxId(stab)
stab$Lambda[ids[1], 1]
stab$params$pi_list[ids[2]]

# Alternative formulation
ids2 <- ArgmaxId(S = stab$S_2d)

# Link with Argmax() function
args <- Argmax(stab)
```

---

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,
  ...
)

```

**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 Y. 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 <a href="#">set.seed</a> ).   |
| 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 <a href="#">mclapply</a> ). Only available on Unix systems.   |

|                          |  |
|--------------------------|--|
| <code>output_data</code> | logical indicating if the input datasets <code>xdata</code> and <code>ydata</code> should be included in the output. |
| <code>verbose</code>     | logical indicating if a loading bar and messages should be printed.  |
| <code>...</code>         | 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 ( $\text{LambdaX}$ ,  $\text{LambdaY}$ ,  $\text{AlphaX}$ , and/or  $\text{AlphaY}$ ). 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  $\lambda$ ) 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](https://doi.org/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](https://doi.org/10.1111/j.14679868.2010.00740.x).
- Liquet B, de Micheaux PL, Hejblum BP, Thiébaud 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](https://doi.org/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](https://doi.org/10.2202/15446115.1390).
- 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](https://doi.org/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](https://doi.org/10.1016/j.jmva.2007.06.007).

## See Also

[SparsePCA](#), [SparsePLS](#), [GroupPLS](#), [SparseGroupPLS](#), [VariableSelection](#), [Resample](#), [StabilityScore](#)  
 Other stability selection functions: [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$data,
  ncomp = 3,
  LambdaX = 1:(ncol(simul$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 = 50, pk = c(5, 5, 5), 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)

# sgPLS: sparsity on both X and Y
```

```

stab <- BiSelection(
  xdata = x, ydata = y, K = 10,
  group_x = c(2, 8, 5), group_y = c(1, 2),
  family = "gaussian", ncomp = 3,
  LambdaX = 1:2, AlphaX = seq(0.1, 0.9, by = 0.2),
  LambdaY = 1:2, AlphaY = seq(0.1, 0.9, by = 0.2),
  implementation = SparseGroupPLS,
  n_cat = 2
)
CalibrationPlot(stab)
CalibrationPlot(stab,
  params = c("nx", "alphax", "ny", "alphay")
)
summary(stab)

# gPLS: sparsity on X
stab <- BiSelection(
  xdata = x, ydata = y,
  group_x = c(2, 8, 5),
  family = "gaussian", ncomp = 3,
  LambdaX = 1:2,
  implementation = GroupPLS
)
CalibrationPlot(stab)
summary(stab)

# gPLS: sparsity on both X and Y
stab <- BiSelection(
  xdata = x, ydata = y,
  group_x = c(2, 8, 5), group_y = c(1, 2),
  family = "gaussian", ncomp = 3,
  LambdaX = 1:2, LambdaY = 1:2,
  implementation = GroupPLS
)
CalibrationPlot(stab)
summary(stab)

## Sparse/Group PLS-DA (Discriminant Analysis)

# Data simulation (categorical outcomes)
set.seed(1)
simul <- SimulateRegression(n = 200, pk = c(5, 5, 5), family = "binomial")
x <- simul$xdata
y <- simul$ydata

# sPLS-DA: sparsity on X
stab <- BiSelection(
  xdata = x, ydata = cbind(y),
  family = "binomial", ncomp = 3,
  LambdaX = 1:(ncol(x) - 1),
  implementation = SparsePLS
)

```

```
CalibrationPlot(stab)
summary(stab)

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

# gPLS-DA: sparsity on X
stab <- BiSelection(
  xdata = x, ydata = cbind(y),
  group_x = c(2, 8, 5),
  family = "binomial", ncomp = 3,
  LambdaX = 1:2,
  implementation = GroupPLS
)
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)
```

**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.

**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,
  xlab = NULL,
  ylab = NULL,
  zlab = expression(italic(q)),
  xlas = 2,
  ylas = NULL,
  zlas = 2,
  cex.lab = 1.5,
  cex.axis = 1,
  xgrid = FALSE,
  ygrid = FALSE,
  params = c("ny", "alphay", "nx", "alphax")
)

```

**Arguments**

|                        |  |
|------------------------|--|
| <code>stability</code> | output of <a href="#">VariableSelection</a> , <a href="#">GraphicalModel</a> or <a href="#">BiSelection</a> .  |
| <code>block_id</code>  | ID of the block to visualise. Only used for multi-block stability selection graphical models. If <code>block_id=NULL</code> , all blocks are represented in separate panels. |
| <code>col</code>       | vector of colours.   |
| <code>pch</code>       | type of point, as in <a href="#">points</a> .  |
| <code>cex</code>       | size of point.   |
| <code>xlim</code>      | displayed range along the x-axis. Only used if <code>stability</code> is the output of <a href="#">BiSelection</a> .   |
| <code>ylim</code>      | displayed range along the y-axis. Only used if <code>stability</code> is the output of <a href="#">BiSelection</a> .   |
| <code>bty</code>       | 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 <a href="#">BiSelection</a> .   |
| lty           | line type, as in <a href="#">par</a> . Only used if stability is the output of <a href="#">BiSelection</a> .  |
| lwd           | line width, as in <a href="#">par</a> . Only used if stability is the output of <a href="#">BiSelection</a> .   |
| 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 <a href="#">BiSelection</a> .   |
| 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 <a href="#">BiSelection</a> with penalisation of the outcomes.                    |
| offset        | distance between the point and the text, as in <a href="#">text</a> . 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 <a href="#">VariableSelection</a> or <a href="#">GraphicalModel</a> .   |
| legend_range  | range of the colour bar. Only used if stability is the output of <a href="#">VariableSelection</a> or <a href="#">GraphicalModel</a> .  |
| 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 <a href="#">VariableSelection</a> or <a href="#">GraphicalModel</a> .  |
| xlas          | orientation of labels on the x-axis, as las in <a href="#">par</a> .  |
| ylas          | orientation of labels on the y-axis, as las in <a href="#">par</a> .  |
| zlas          | orientation of labels on the z-axis, as las in <a href="#">par</a> .  |
| cex.lab       | font size for labels.   |
| cex.axis      | font size for axes.   |
| xgrid         | logical indicating if a vertical grid should be drawn. Only used if stability is the output of <a href="#">BiSelection</a> .  |
| ygrid         | logical indicating if a horizontal grid should be drawn. Only used if stability is the output of <a href="#">BiSelection</a> .  |
| params        | vector of possible parameters if stability is of class <code>bi_selection</code> . The order of these parameters defines the order in which they are represented. Only used if stability is the output of <a href="#">BiSelection</a> . |

**Value**

a calibration plot.

**See Also**

[VariableSelection](#), [GraphicalModel](#), [BiSelection](#)

Other calibration functions: [Adjacency\(\)](#), [ArgmaxId\(\)](#), [Argmax\(\)](#), [SelectedVariables\(\)](#), [SelectionProportions\(\)](#)

**Examples**

```
oldpar <- par(no.readonly = TRUE)
par(mar = c(7, 5, 7, 6))

## Regression model

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

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

# Calibration heatmap
CalibrationPlot(stab)

# User-defined colours
CalibrationPlot(stab,
  col = c("ivory", "blue", "black"),
  legend_length = 31,
  legend_range = c(0, 2500)
)

## Dimensionality reduction

# Data simulation
set.seed(1)
simul <- SimulateRegression(n = 50, pk = c(5, 5, 5), family = "gaussian")
x <- simul$xdata
y <- simul$ydata

# 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
)

# Calibration plot
CalibrationPlot(stab)

# Other ordering of parameters
CalibrationPlot(stab, params = c("nx", "ny"))

par(oldpar)
```

---

Combine

*Merging stability selection outputs*

---

## Description

Merges the outputs from [VariableSelection](#) or [GraphicalModel](#). 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

|                           |  |
|---------------------------|--|
| <code>stability1</code>   | output from a first run of <a href="#">VariableSelection</a> or <a href="#">GraphicalModel</a> .                             |
| <code>stability2</code>   | output from a second run of <a href="#">VariableSelection</a> or <a href="#">GraphicalModel</a> .                            |
| <code>include_beta</code> | logical indicating if the beta coefficients of visited models should be concatenated. Only applicable to variable selection. |

## 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)
print(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)
print(stab)

```

---

Ensemble

*Ensemble model*


---

## Description

Creates an ensemble predictive model from [VariableSelection](#) outputs.

## Usage

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

## Arguments

|           |  |
|-----------|--|
| stability | output of <a href="#">VariableSelection</a> .  |
| 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

|                       |  |
|-----------------------|--|
| <code>ensemble</code> | output of <a href="#">Ensemble</a> .                                     |
| <code>xdata</code>    | matrix of predictors with observations as rows and variables as columns. |
| <code>...</code>      | additional parameters passed to <a href="#">predict</a> .                |

## Value

A matrix of predictions computed from the observations in `xdata`.

## See Also

Other ensemble model functions: [Ensemble\(\)](#)

**Examples**

```

# Linear regression
set.seed(1)
simul <- SimulateRegression(n = 1000, pk = 50, family = "gaussian")
ids <- Split(data = simul$ydata, tau = c(0.8, 0.2))
stab <- VariableSelection(
  xdata = simul$xdata[ids[[1]], ],
  ydata = simul$ydata[ids[[1]], ]
)
ensemble <- Ensemble(
  stability = stab,
  xdata = simul$xdata[ids[[1]], ],
  ydata = simul$ydata[ids[[1]], ]
)
yhat <- EnsemblePredictions(
  ensemble = ensemble,
  xdata = simul$xdata[ids[[2]], ]
)
cor(simul$ydata[ids[[2]], ], yhat)^2 # Q-squared

# Logistic regression
set.seed(1)
simul <- SimulateRegression(n = 1000, pk = 20, family = "binomial", ev_xz = 0.9)
ids <- Split(data = simul$ydata, family = "binomial", tau = c(0.8, 0.2))
stab <- VariableSelection(
  xdata = simul$xdata[ids[[1]], ],
  ydata = simul$ydata[ids[[1]], ],
  family = "binomial"
)
ensemble <- Ensemble(
  stability = stab,
  xdata = simul$xdata[ids[[1]], ],
  ydata = simul$ydata[ids[[1]], ]
)
yhat <- EnsemblePredictions(
  ensemble = ensemble,
  xdata = simul$xdata[ids[[2]], ]
)
PlotROC(ROC(predicted = yhat, observed = simul$ydata[ids[[2]], ]))
yhat <- EnsemblePredictions(
  ensemble = ensemble,
  xdata = simul$xdata[ids[[2]], ],
  type = "response"
) # predicted probabilities

```

**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,
  stability = NULL,
  family = NULL,
  implementation = NULL,
  prediction = NULL,
  K = 1,
  tau = 0.8,
  seed = 1,
  n_thr = NULL,
  ij_method = FALSE,
  time = 1000
)
```

**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 <a href="#">VariableSelection</a> . 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 <a href="#">VariableSelection</a> , <a href="#">lm</a> (linear regression), <a href="#">coxph</a> (Cox regression), <a href="#">glm</a> (logistic regression), or <a href="#">multinom</a> (multinomial regression) is used. |
| prediction     | optional function to compute predicted values from the model refitted with implementation.   |
| K              | number of training-test splits.  |
| tau            | proportion of observations used in the training set.   |
| seed           | value of the seed to ensure reproducibility of the results.  |
| 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.  |

|                        |  |
|------------------------|--|
| <code>ij_method</code> | 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 <a href="#">concordance</a> . If <code>ij_method=FALSE</code> (the default), the concordance indices computed for different refitting/test splits are reported. If <code>ij_method=TRUE</code> , the concordance index and estimated confidence interval at level 0.05 are reported. Only applicable to Cox regression. |
| <code>time</code>      | numeric indicating the time for which the survival probabilities are computed. Only applicable to Cox regression.  |

### 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).  |
| <code>concordance</code> | Concordance index (for Cox regression only).  |
| <code>lower</code>       | lower bound of the confidence interval at level 0.05 for the concordance index calculated using the infinitesimal jackknife (for Cox regression and with <code>ij_method=TRUE</code> ). |
| <code>upper</code>       | upper bound of the confidence interval at level 0.05 for the concordance index calculated using the infinitesimal jackknife (for Cox regression and with <code>ij_method=TRUE</code> ). |
| Beta                     | matrix of estimated beta coefficients across the K iterations. Coefficients are extracted using the <a href="#">coef</a> function.  |

### See Also

[VariableSelection](#), [Refit](#)

Other prediction performance functions: [Incremental\(\)](#), [PlotIncremental\(\)](#), [PlotROC\(\)](#), [ROC\(\)](#)

**Examples**

```

## Logistic regression

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

# Balanced split: 50% variable selection set and 50% for evaluation of performances
ids_train <- Resample(
  data = simul$ydata,
  tau = 0.5, family = "binomial"
)
xtrain <- simul$xdata[ids_train, ]
ytrain <- simul$ydata[ids_train, ]
xtest <- simul$xdata[-ids_train, ]
ytest <- simul$ydata[-ids_train, ]

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

# Evaluation of the performances on refitted models (K=1)
roc <- ExplanatoryPerformance(
  xdata = xtest, ydata = ytest,
  stability = stab, n_thr = NULL
)
PlotROC(roc)

# Using more refitting/test splits
roc <- ExplanatoryPerformance(
  xdata = xtest, ydata = ytest,
  stability = stab, K = 100
)
boxplot(roc$AUC, ylab = "AUC")
PlotROC(roc)

# Comparison with saturated model
roc <- ExplanatoryPerformance(
  xdata = xtest, ydata = ytest,
  family = "binomial", K = 100
)
PlotROC(roc, col = "blue", col_band = "blue", add = TRUE)

## Partial Least Squares (single component)

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

```

```

print(SelectedVariables(stab))

# Defining wrapping functions for 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)
}

# Evaluation of the performances on refitted models (K=1)
roc <- ExplanatoryPerformance(
  xdata = xtest, ydata = ytest,
  stability = stab,
  implementation = PLSDA, prediction = PredictPLSDA
)
PlotROC(roc)

## Cox regression

# Data simulation
set.seed(1)
simul <- SimulateRegression(n = 500, pk = 50, family = "binomial")
ydata <- cbind(
  time = runif(nrow(simul$ydata), min = 100, max = 2000),
  case = simul$ydata[, 1]
) # including dummy time to event

# Balanced split: 50% variable selection set and 50% for evaluation of performances
ids_train <- Resample(
  data = simul$ydata,
  tau = 0.5, family = "binomial"
)
xtrain <- simul$xdata[ids_train, ]
ytrain <- ydata[ids_train, ]
xtest <- simul$xdata[-ids_train, ]
ytest <- ydata[-ids_train, ]

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

# Evaluation of the performances on refitted models (K=1)
perf <- ExplanatoryPerformance(
  xdata = xtest, ydata = ytest,
  stability = stab, ij_method = TRUE
)
print(perf)

# Using more refitting/test splits

```

```
perf <- ExplanatoryPerformance(  
  xdata = xtest, ydata = ytest,  
  stability = stab, K = 10, time = 1000  
)  
boxplot(perf$concordance)  
  
## Linear regression  
  
# Data simulation  
set.seed(1)  
simul <- SimulateRegression(n = 1000, pk = 10, family = "gaussian")  
  
# Balanced split: 50% variable selection set and 50% for evaluation of performances  
ids_train <- Resample(  
  data = simul$ydata,  
  tau = 0.5, family = "gaussian"  
)  
xtrain <- simul$xdata[ids_train, ]  
ytrain <- simul$ydata[ids_train, ]  
xtest <- simul$xdata[-ids_train, ]  
ytest <- simul$ydata[-ids_train, ]  
  
# Stability selection  
stab <- VariableSelection(xdata = xtrain, ydata = ytrain, family = "gaussian")  
  
# Evaluation of the performances on refitted models (K=1)  
perf <- ExplanatoryPerformance(  
  xdata = xtest, ydata = ytest,  
  stability = stab  
)  
print(perf)  
  
## Partial Least Squares (single component)  
  
# Stability selection  
stab <- VariableSelection(  
  xdata = xtrain, ydata = ytrain,  
  implementation = SparsePLS,  
  family = "gaussian"  
)  
print(SelectedVariables(stab))  
  
# Evaluation of the performances on refitted models (K=1)  
perf <- ExplanatoryPerformance(  
  xdata = xtest, ydata = ytest,  
  stability = stab,  
  implementation = PLS, prediction = PredictPLS  
)  
print(perf)
```

---

|     |                                   |
|-----|-----------------------------------|
| FDP | <i>False Discovery Proportion</i> |
|-----|-----------------------------------|

---

**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: [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 | <i>Splitting observations into folds</i> |
|-------|--|

---

**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 <a href="#">glmnet</a> . 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, ])
})
```

---

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 <a href="#">GraphicalModel</a> .  |
| 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 <a href="#">graph_from_adjacency_matrix</a> ).  |
| 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$xdata)
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
  igrph::V(vgraph)$shape <- rep("dot", length(igrph::V(vgraph)))
  v <- visNetwork::visIgraph(vgraph)
  mylayout <- as.matrix(v$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 <a href="#">igraph</a> object, or output of <a href="#">GraphicalModel</a> , <a href="#">VariableSelection</a> , <a href="#">BiSelection</a> , or output of <a href="#">SimulateGraphical</a> , <a href="#">SimulateRegression</a> . |
| 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, <a href="#">LambdaGridGraphical</a> 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 <a href="#">glassoFast</a> 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 <a href="#">glassoFast</a> 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 <a href="#">glassoFast</a> ).   |
| ...                 | 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,
    ...
)

```

### Arguments

|                                  |  |
|----------------------------------|--|
| <code>xdata</code>               | 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.  |
| <code>pk</code>                  | optional vector encoding the grouping structure. Only used for multi-block stability selection where <code>pk</code> indicates the number of variables in each group. If <code>pk=NULL</code> , single-block stability selection is performed.   |
| <code>Lambda</code>              | matrix of parameters controlling the level of sparsity in the underlying feature selection algorithm specified in <code>implementation</code> . If <code>Lambda=NULL</code> and <code>implementation=PenalisedGraphical</code> , <code>LambdaGridGraphical</code> is used to define a relevant grid. <code>Lambda</code> can be provided as a vector or a matrix with <code>length(pk)</code> columns.   |
| <code>lambda_other_blocks</code> | 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, <code>lambda_other_blocks</code> must be set to <code>NULL</code> (not recommended). Only used for multi-block stability selection, i.e. if <code>length(pk)&gt;1</code> .  |
| <code>pi_list</code>             | vector of thresholds in selection proportions. If <code>n_cat=3</code> , these values must be $>0.5$ and $<1$ . If <code>n_cat=2</code> , these values must be $>0$ and $<1$ .   |
| <code>K</code>                   | number of resampling iterations.   |
| <code>tau</code>                 | subsample size. Only used if <code>resampling="subsampling"</code> and <code>cpss=FALSE</code> .   |
| <code>seed</code>                | value of the seed to initialise the random number generator and ensure reproducibility of the results (see <a href="#">set.seed</a> ).   |
| <code>n_cat</code>               | number of categories used to compute the stability score. Possible values are 2 or 3.  |
| <code>implementation</code>      | function to use for graphical modelling. If <code>implementation=PenalisedGraphical</code> , the algorithm implemented in <code>glassoFast</code> is used for regularised estimation of a conditional independence graph. Alternatively, a user-defined function can be provided.  |
| <code>start</code>               | character string indicating if the algorithm should be initialised at the estimated (inverse) covariance with previous penalty parameters ( <code>start="warm"</code> ) or not ( <code>start="cold"</code> ). Using <code>start="warm"</code> can speed-up the computations, but could lead to convergence issues (in particular with small <code>Lambda_cardinal</code> ). Only used for <code>implementation=PenalisedGraphical</code> (see argument "start" in <code>glassoFast</code> ). |
| <code>scale</code>               | logical indicating if the correlation ( <code>scale=TRUE</code> ) or covariance ( <code>scale=FALSE</code> ) matrix should be used as input of <code>glassoFast</code> if <code>implementation=PenalisedGraphical</code> . Otherwise, this argument must be used in the function provided in <code>implementation</code> .   |
| <code>resampling</code>          | resampling approach. Possible values are: "subsampling" for sampling without replacement of a proportion <code>tau</code> 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 <code>data</code> and <code>tau</code> and return the IDs of observations to be included in the resampled dataset.   |
| <code>cpss</code>               | 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 <code>PFER_method="MB"</code> . |
| <code>PFER_method</code>        | method used to compute the upper-bound of the expected number of False Positives (or Per Family Error Rate, PFER). If <code>PFER_method="MB"</code> , the method proposed by Meinshausen and Bühlmann (2010) is used. If <code>PFER_method="SS"</code> , the method proposed by Shah and Samworth (2013) under the assumption of unimodality is used.                         |
| <code>PFER_thr</code>           | threshold in PFER for constrained calibration by error control. If <code>PFER_thr=Inf</code> and <code>FDP_thr=Inf</code> , unconstrained calibration is used (the default).  |
| <code>FDP_thr</code>            | threshold in the expected proportion of falsely selected features (or False Discovery Proportion) for constrained calibration by error control. If <code>PFER_thr=Inf</code> and <code>FDP_thr=Inf</code> , unconstrained calibration is used (the default).  |
| <code>Lambda_cardinal</code>    | number of values in the grid of parameters controlling the level of sparsity in the underlying algorithm. Only used if <code>Lambda=NULL</code> .   |
| <code>lambda_max</code>         | optional maximum value for the grid in penalty parameters. If <code>lambda_max=NULL</code> , the maximum value is set to the maximum covariance in absolute value. Only used if <code>implementation=PenalisedGraphical</code> and <code>Lambda=NULL</code> .   |
| <code>lambda_path_factor</code> | multiplicative factor used to define the minimum value in the grid.   |
| <code>max_density</code>        | threshold on the density. The grid is defined such that the density of the estimated graph does not exceed <code>max_density</code> .   |
| <code>n_cores</code>            | number of cores to use for parallel computing (see <a href="#">mclapply</a> ). Only available on Unix systems.  |
| <code>output_data</code>        | logical indicating if the input datasets <code>xdata</code> and <code>ydata</code> should be included in the output.  |
| <code>verbose</code>            | logical indicating if a loading bar and messages should be printed.   |
| <code>...</code>                | additional parameters passed to the functions provided in <code>implementation</code> or <code>resampling</code> .  |

## 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:

|                     |   |
|---------------------|---|
| <code>S</code>      | a matrix of the best stability scores for different (sets of) parameters controlling the level of sparsity in the underlying algorithm.               |
| <code>Lambda</code> | a matrix of parameters controlling the level of sparsity in the underlying algorithm.   |
| <code>Q</code>      | a matrix of the average number of selected features by the underlying algorithm with different parameters controlling the level of sparsity.          |
| <code>Q_s</code>    | a matrix of the calibrated number of stably selected features with different parameters controlling the level of sparsity.                            |
| <code>P</code>      | a matrix of calibrated thresholds in selection proportions for different parameters controlling the level of sparsity in the underlying algorithm.    |
| <code>PFER</code>   | a matrix of upper-bounds in <code>PFER</code> of calibrated stability selection models with different parameters controlling the level of sparsity.   |
| <code>FDP</code>    | a matrix of upper-bounds in <code>FDP</code> of calibrated stability selection models with different parameters controlling the level of sparsity.    |
| <code>S_2d</code>   | 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 <code>length(pk)=1</code> .                     |
| 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 <code>length(pk)=1</code> .                    |
| 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 <code>xdata</code> .   |
| method  | a list with <code>type="graphical_model"</code> 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 <code>xdata</code> ), 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>.
- 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](https://doi.org/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](https://doi.org/10.1111/j.14679868.2011.01034.x).

## See Also

[PenalisedGraphical](#), [GraphicalAlgo](#), [LambdaGridGraphical](#), [Resample](#), [StabilityScore Graph](#), [Adjacency](#),

Other stability selection functions: [BiSelection\(\)](#), [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)

```

**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 <a href="#">gPLS</a> is run (for continuous outcomes). If family="binomial", a PLS-DA model as defined in <a href="#">gPLSda</a> 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 <a href="#">sgPLS</a> is obtained by concatenating keepX_previous and Lambda.  |
| keepY          | number of selected groups of outcome variables. This argument is defined as in <a href="#">sgPLS</a> . 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 <a href="#">gPLS</a> or <a href="#">gPLSda</a> .  |

**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, Thiébaud 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](https://doi.org/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 = c(10, 20, 20), 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
)
```

)

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,
  stability = NULL,
  family = NULL,
  implementation = NULL,
  prediction = NULL,
  n_predictors = NULL,
  K = 100,
  tau = 0.8,
  seed = 1,
  n_thr = NULL,
  ij_method = FALSE,
  time = 1000
)
```

**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 <a href="#">VariableSelection</a> . 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 <a href="#">VariableSelection</a> , <a href="#">lm</a> (linear regression), <a href="#">coxph</a> (Cox regression), <a href="#">glm</a> (logistic regression), or <a href="#">multinom</a> (multinomial regression) is used. |

|              |   |
|--------------|---|
| prediction   | optional function to compute predicted values from the model refitted with implementation.  |
| n_predictors | number of predictors to consider.   |
| K            | number of training-test splits.   |
| tau          | proportion of observations used in the training set.  |
| seed         | value of the seed to ensure reproducibility of the results.   |
| 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 <a href="#">concordance</a> . 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.   |

### 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.   |

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.

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.

### See Also

[VariableSelection](#), [Refit](#)

Other prediction performance functions: [ExplanatoryPerformance\(\)](#), [PlotIncremental\(\)](#), [PlotROC\(\)](#), [ROC\(\)](#)

### Examples

```
## Logistic regression

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

# Balanced split: 50% variable selection set and 50% for evaluation of performances
ids_train <- Resample(
  data = simul$ydata,
  tau = 0.5, family = "binomial"
)
xtrain <- simul$xdata[ids_train, ]
ytrain <- simul$ydata[ids_train, ]
xtest <- simul$xdata[-ids_train, ]
ytest <- simul$ydata[-ids_train, ]

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

# Evaluating marginal contribution of the predictors
perf <- Incremental(xdata = xtest, ydata = ytest, stability = stab, K = 10)
summary(perf)

# Visualisation
PlotIncremental(perf)
plot(perf) # alternative formulation

## Partial Least Squares (single component)

# Stability selection
stab <- VariableSelection(
  xdata = xtrain, ydata = ytrain,
  implementation = SparsePLS,
```

```

    family = "binomial"
  )
print(SelectedVariables(stab))

# Defining wrapping functions for 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)
}

# Evaluation of the performances on refitted models (K=1)
incremental <- Incremental(
  xdata = xtest, ydata = ytest,
  stability = stab,
  implementation = PLSDA, prediction = PredictPLSDA,
  K = 10
)
PlotIncremental(incremental)

## Cox regression

# Data simulation
set.seed(1)
simul <- SimulateRegression(n = 1000, pk = 50, family = "binomial")
ydata <- cbind(
  time = runif(nrow(simul$ydata), min = 100, max = 2000),
  case = simul$ydata[, 1]
) # including dummy time to event

# Balanced split: 50% variable selection set and 50% for evaluation of performances
ids_train <- Resample(
  data = simul$ydata,
  tau = 0.5, family = "binomial"
)
xtrain <- simul$xdata[ids_train, ]
ytrain <- ydata[ids_train, ]
xtest <- simul$xdata[-ids_train, ]
ytest <- ydata[-ids_train, ]

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

# Marginal contribution
perf <- Incremental(xdata = xtest, ydata = ytest, stability = stab, K = 10)
PlotIncremental(perf)

# Faster computations on a single data split

```

```
perf <- Incremental(xdata = xtest, ydata = ytest, stability = stab, ij_method = TRUE)
PlotIncremental(perf)

## Linear regression

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

# Balanced split: 50% variable selection set and 50% for evaluation of performances
ids_train <- Resample(
  data = simul$ydata,
  tau = 0.5, family = "gaussian"
)
xtrain <- simul$xdata[ids_train, ]
ytrain <- simul$ydata[ids_train, ]
xtest <- simul$xdata[-ids_train, ]
ytest <- simul$ydata[-ids_train, ]

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

# Evaluating marginal contribution of the predictors
perf <- Incremental(xdata = xtest, ydata = ytest, stability = stab, K = 10)
PlotIncremental(perf)

## Partial Least Squares (single component)

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

# Evaluation of the performances on refitted models (K=1)
incremental <- Incremental(
  xdata = xtest, ydata = ytest,
  stability = stab,
  implementation = PLS, prediction = PredictPLS,
  K = 10
)
PlotIncremental(incremental)
```

**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**

|                                  |   |
|----------------------------------|---|
| <code>xdata</code>               | 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.   |
| <code>pk</code>                  | optional vector encoding the grouping structure. Only used for multi-block stability selection where <code>pk</code> indicates the number of variables in each group. If <code>pk=NULL</code> , single-block stability selection is performed.  |
| <code>lambda_other_blocks</code> | 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, <code>lambda_other_blocks</code> must be set to <code>NULL</code> (not recommended). Only used for multi-block stability selection, i.e. if <code>length(pk)&gt;1</code> . |
| <code>K</code>                   | number of resampling iterations.  |
| <code>tau</code>                 | subsample size. Only used if <code>resampling="subsampling"</code> and <code>cpss=FALSE</code> .  |
| <code>n_cat</code>               | number of categories used to compute the stability score. Possible values are 2 or 3.   |
| <code>implementation</code>      | function to use for graphical modelling. If <code>implementation=PenalisedGraphical</code> , the algorithm implemented in <a href="#">glassoFast</a> 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 <a href="#">glassoFast</a> ).  |
| scale              | logical indicating if the correlation (scale=TRUE) or covariance (scale=FALSE) matrix should be used as input of <a href="#">glassoFast</a> 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

```

**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 <a href="#">set.seed</a> ).   |
| family          | type of regression model. This argument is defined as in <a href="#">glmnet</a> . 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)
```

---

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 <a href="#">glassoFast</a> 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: [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 <a href="#">glmnet</a> . Possible values include "gaussian" (linear regression), "binomial" (logistic regression), "multinomial" (multinomial regression), and "cox" (survival analysis). |
| ...    | additional parameters passed to <a href="#">glmnet</a> .   |

**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: [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](https://doi.org/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](https://doi.org/10.1111/j.14679868.2011.01034.x).

**See Also**

Other stability metric functions: [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")
```

## 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.

## Usage

```
PlotIncremental(  
  perf,  
  quantiles = c(0.05, 0.95),  
  ylab = "Performance",  
  pch = 18,  
  col = "black",  
  col.axis = NULL,  
  cex = 1,  
  cex.lab = 1.5,  
  xcex.axis = 1,  
  ycex.axis = 1,  
  xlas = 2,  
  ylas = 1,  
  sfrac = 0.005,  
  ylim = NULL,  
  bty = "o",  
  xgrid = FALSE,  
  ygrid = FALSE,  
  output_data = FALSE  
)
```

## Arguments

|           |  |
|-----------|--|
| perf      | output of <a href="#">Incremental</a> .  |
| quantiles | quantiles defining the lower and upper bounds.   |
| ylab      | label of the y-axis.   |
| pch       | type of point, as in <a href="#">points</a> .  |
| col       | vector of point colours.   |
| col.axis  | optional vector of label colours. If col.axis=NULL, the colours provided in argument col are used. |
| cex       | size of point.   |
| cex.lab   | font size for labels.  |
| xcex.axis | size of labels along the x-axis.   |

|                          |  |
|--------------------------|--|
| <code>ycex.axis</code>   | size of labels along the y-axis.   |
| <code>xlas</code>        | orientation of labels on the x-axis, as <code>las</code> in <a href="#">par</a> .  |
| <code>ylas</code>        | orientation of labels on the y-axis, as <code>las</code> in <a href="#">par</a> .  |
| <code>sfrac</code>       | size of the end bars, as in <a href="#">plotCI</a> .   |
| <code>ylim</code>        | displayed range along the y-axis. Only used if <code>stability</code> is the output of <a href="#">BiSelection</a> .                               |
| <code>bty</code>         | 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). |
| <code>xgrid</code>       | logical indicating if a vertical grid should be drawn. Only used if <code>stability</code> is the output of <a href="#">BiSelection</a> .          |
| <code>ygrid</code>       | logical indicating if a horizontal grid should be drawn. Only used if <code>stability</code> is the output of <a href="#">BiSelection</a> .        |
| <code>output_data</code> | logical indicating if the median and quantiles should be returned in a matrix.   |

**Value**

A plot.

**See Also**

[VariableSelection](#), [Refit](#)

Other prediction performance functions: [ExplanatoryPerformance\(\)](#), [Incremental\(\)](#), [PlotROC\(\)](#), [ROC\(\)](#)

**Examples**

```
## Logistic regression

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

# Balanced split: 50% variable selection set and 50% for evaluation of performances
ids_train <- Resample(
  data = simul$ydata,
  tau = 0.5, family = "binomial"
)
xtrain <- simul$xdata[ids_train, ]
ytrain <- simul$ydata[ids_train, ]
xtest <- simul$xdata[-ids_train, ]
ytest <- simul$ydata[-ids_train, ]

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

# Evaluating marginal contribution of the predictors
perf <- Incremental(xdata = xtest, ydata = ytest, stability = stab, K = 10)
```

```
# Basic visualisation
PlotIncremental(perf)

# Adding grids
PlotIncremental(perf, xgrid = TRUE, ygrid = TRUE)

# Changing colours
PlotIncremental(perf,
  bty = "n",
  col = colorRampPalette(c("blue", "red"))(length(perf$names))
)

## Cox regression

# Data simulation
set.seed(1)
simul <- SimulateRegression(n = 1000, pk = 50, family = "binomial")
ydata <- cbind(
  time = runif(nrow(simul$ydata), min = 100, max = 2000),
  case = simul$ydata[, 1]
) # including dummy time to event

# Balanced split: 50% variable selection set and 50% for evaluation of performances
ids_train <- Resample(
  data = simul$ydata,
  tau = 0.5, family = "binomial"
)
xtrain <- simul$xdata[ids_train, ]
ytrain <- ydata[ids_train, ]
xtest <- simul$xdata[-ids_train, ]
ytest <- ydata[-ids_train, ]

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

# Marginal contribution
perf <- Incremental(xdata = xtest, ydata = ytest, stability = stab, K = 10)
PlotIncremental(perf)

# Faster computations on a single data split
perf <- Incremental(xdata = xtest, ydata = ytest, stability = stab, ij_method = TRUE)
PlotIncremental(perf)

## Linear regression

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

# Balanced split: 50% variable selection set and 50% for evaluation of performances
```

```

ids_train <- Resample(
  data = simul$ydata,
  tau = 0.5, family = "gaussian"
)
xtrain <- simul$xdata[ids_train, ]
ytrain <- simul$ydata[ids_train, ]
xtest <- simul$xdata[-ids_train, ]
ytest <- simul$ydata[-ids_train, ]

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

# Evaluating marginal contribution of the predictors
perf <- Incremental(xdata = xtest, ydata = ytest, stability = stab, K = 10)
PlotIncremental(perf)

# Evaluating marginal contribution of the predictors beyond stably selected
perf <- Incremental(
  xdata = xtest, ydata = ytest, stability = stab,
  K = 10, n_predictors = 20
)
N <- sum(SelectedVariables(stab))
PlotIncremental(perf, col = c(rep("red", N), rep("grey", 20 - N)))

```

---

PlotROC

---

*Receiver Operating Characteristic (ROC) curve*


---

### 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.

### Usage

```

PlotROC(
  roc,
  xlab = "False Positive Rate",
  ylab = "True Positive Rate",
  col = "red",
  col_band = NULL,
  alpha = 0.5,
  lwd = 1,
  lty = 1,
  quantiles = c(0.05, 0.95),
  add = FALSE
)

```

**Arguments**

|           |  |
|-----------|--|
| roc       | output of <a href="#">ROC</a> or <a href="#">ExplanatoryPerformance</a> .  |
| xlab      | label of the x-axis.   |
| ylab      | label of the y-axis.   |
| col       | colour of the point-wise median curve.   |
| col_band  | colour of the band defined by point-wise quantiles.  |
| alpha     | level of opacity for the band.   |
| lwd       | line width, as in <a href="#">par</a> . Only used if <code>stability</code> is the output of <a href="#">BiSelection</a> . |
| lty       | line type, as in <a href="#">par</a> . Only used if <code>stability</code> is the output of <a href="#">BiSelection</a> .  |
| quantiles | point-wise quantiles of the performances defining the band.  |
| add       | logical indicating if the curve should be added to the current plot.   |

**Value**

A plot.

**See Also**

[VariableSelection](#), [Refit](#)

Other prediction performance functions: [ExplanatoryPerformance\(\)](#), [Incremental\(\)](#), [PlotIncremental\(\)](#), [ROC\(\)](#)

**Examples**

```
# Data simulation
set.seed(1)
simul <- SimulateRegression(n = 500, pk = 10, family = "binomial")

# Balanced split: 50% variable selection set and 50% for evaluation of performances
ids_train <- Resample(
  data = simul$ydata,
  tau = 0.5, family = "binomial"
)
xtrain <- simul$xdata[ids_train, ]
ytrain <- simul$ydata[ids_train, ]
xtest <- simul$xdata[-ids_train, ]
ytest <- simul$ydata[-ids_train, ]

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

# Evaluation of the performances on refitted models (K=1)
roc <- ExplanatoryPerformance(
  xdata = xtest, ydata = ytest,
  stability = stab, n_thr = NULL
)
PlotROC(roc)
```

```

# Using more refitting/test splits
roc <- ExplanatoryPerformance(
  xdata = xtest, ydata = ytest,
  stability = stab, K = 100
)
PlotROC(roc)

# Comparison with saturated model
roc <- ExplanatoryPerformance(
  xdata = xtest, ydata = ytest,
  family = "binomial", K = 100
)
PlotROC(roc, col = "blue", col_band = "blue", add = TRUE)

```

---

 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 $(n\text{col}(x\text{data}) * n\text{comp})$ . 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 $(n\text{col}(y\text{data}) * n\text{comp})$ . 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 <a href="#">pls</a> (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  $W_{mat}$  is the equivalent of `loadings$X` in [pls](#). The loadings matrix  $P_{mat}$  is the equivalent of `mat.c` in [pls](#). The score matrices  $T_{mat}$  and  $Q_{mat}$  are the equivalent of `variates$X` and `variates$Y` in [pls](#).

### Value

A list with:

|                      |  |
|----------------------|--|
| $W_{mat}$            | matrix of X-weights.   |
| $W_{star}$           | matrix of transformed X-weights.   |
| $P_{mat}$            | matrix of X-loadings.  |
| $C_{mat}$            | matrix of Y-weights.   |
| $T_{mat}$            | matrix of X-scores.  |
| $U_{mat}$            | matrix of Y-scores.  |
| $Q_{mat}$            | matrix needed for predictions.   |
| $R_{mat}$            | matrix needed for predictions.   |
| <code>meansX</code>  | vector used for centering of predictors, needed for predictions.                       |
| <code>sigmaX</code>  | vector used for scaling of predictors, needed for predictions.                         |
| <code>meansY</code>  | vector used for centering of outcomes, needed for predictions.                         |
| <code>sigmaY</code>  | vector used for scaling of outcomes, needed for predictions.                           |
| <code>methods</code> | a list with <code>family</code> and <code>scale</code> values used for the run.        |
| <code>params</code>  | a list with <code>selectedX</code> and <code>selectedY</code> 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](https://doi.org/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 = c(5, 5, 5), 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)
```

---

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

|                    |  |
|--------------------|--|
| <code>xdata</code> | matrix of predictors with observations as rows and variables as columns. |
| <code>model</code> | output of <a href="#">PLS</a> .  |

### 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,
  ...
)
```

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

**Arguments**

|                             |  |
|-----------------------------|--|
| <code>xdata</code>          | matrix of predictors with observations as rows and variables as columns.   |
| <code>ydata</code>          | optional vector or matrix of outcome(s). If <code>family</code> is set to "binomial" or "multinomial", <code>ydata</code> can be a vector with character/numeric values or a factor.   |
| <code>stability</code>      | output of <a href="#">VariableSelection</a> or <a href="#">BiSelection</a> . If <code>stability=NULL</code> (the default), a model including all variables in <code>xdata</code> as predictors is fitted. Argument <code>family</code> must be provided in this case.  |
| <code>family</code>         | 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 <code>stability</code> .   |
| <code>implementation</code> | optional function to refit the model. If <code>implementation=NULL</code> and <code>stability</code> is the output of <a href="#">VariableSelection</a> , <code>lm</code> (linear regression), <code>coxph</code> (Cox regression), <code>glm</code> (logistic regression), or <code>multinom</code> (multinomial regression) is used. The function <code>PLS</code> is used for the output of <a href="#">BiSelection</a> . |
| <code>...</code>            | additional arguments to be passed to the function provided in <code>implementation</code> .  |

**Value**

The output as obtained from:

|                       |  |
|-----------------------|--|
| <code>lm</code>       | for linear regression ("gaussian" family).         |
| <code>coxph</code>    | for Cox regression ("cox" family).                 |
| <code>glm</code>      | for logistic regression ("binomial" family).       |
| <code>multinom</code> | 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
```

```

## Cox regression

# Data simulation
set.seed(1)
simul <- SimulateRegression(n = 100, pk = 50, family = "binomial")
ydata <- cbind(
  time = runif(nrow(simul$ydata), min = 100, max = 2000),
  case = simul$ydata[, 1]
) # including dummy time to event

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

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

# Refitting the model
refitted <- Refit(
  xdata = xrefit, ydata = yrefit,
  stability = stab
)
refitted$coefficients # refitted coefficients
head(refitted$linear.predictors) # refitted scores

## 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

## Multinomial regression

# Data simulation
set.seed(1)
simul <- SimulateRegression(n = 200, pk = 15, family = "multinomial")

# Data split
ids_train <- Resample(
  data = simul$ydata,
  tau = 0.5, family = "multinomial"
)
xtrain <- simul$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 = "multinomial"
)

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

## Partial Least Squares (single component)

# 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,
  implementation = SparsePLS,
  family = "gaussian"
)
print(SelectedVariables(stab))

# Refitting the model
refitted <- Refit(
  xdata = xrefit, ydata = yrefit,
  implementation = PLS,
  stability = stab
)
refitted$Wmat # refitted X-weights
head(refitted$Tmat) # refitted X-scores

## Partial Least Squares (multiple components)

# Data simulation
set.seed(1)
simul <- SimulateRegression(n = 200, pk = c(5, 5, 5), 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

|                         |   |
|-------------------------|---|
| <code>data</code>       | vector or matrix of data. In regression, this should be the outcome data.   |
| <code>family</code>     | type of regression model. This argument is defined as in <a href="#">glmnet</a> . Possible values include "gaussian" (linear regression), "binomial" (logistic regression), "multinomial" (multinomial regression), and "cox" (survival analysis).  |
| <code>tau</code>        | subsample size. Only used if <code>resampling="subsampling"</code> and <code>cpss=FALSE</code> .  |
| <code>resampling</code> | resampling approach. Possible values are: "subsampling" for sampling without replacement of a proportion <code>tau</code> 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 <code>data</code> and <code>tau</code> and return the IDs of observations to be included in the resampled dataset. |
| <code>...</code>        | additional parameters passed to the function provided in <code>resampling</code> .  |

### 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
)

```

### Description

Computes the True and False Positive Rates (TPR and FPR, respectively) and Area Under the Curve (AUC) by comparing the true (observed) and predicted status using a range of thresholds on the predicted score.

**Usage**

```
ROC(predicted, observed, n_thr = NULL)
```

**Arguments**

|           |   |
|-----------|---|
| predicted | numeric predicted scores.   |
| observed  | factor encoding the observed binary status.   |
| n_thr     | number of thresholds to use to construct the ROC curve. For faster computations on large data, values below <code>length(x)-1</code> can be used. |

**Value**

A list with:

|     |                       |
|-----|-----------------------|
| TPR | True Positive Rate.   |
| FPR | False Positive Rate.  |
| AUC | Area Under the Curve. |

**See Also**

Other prediction performance functions: [ExplanatoryPerformance\(\)](#), [Incremental\(\)](#), [PlotIncremental\(\)](#), [PlotROC\(\)](#)

**Examples**

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

# Balanced training/test split
ids_train <- Resample(
  data = simul$ydata,
  tau = 0.5, family = "binomial"
)
xtrain <- simul$xdata[ids_train, , drop = FALSE]
ytrain <- simul$ydata[ids_train, , drop = FALSE]
x2 <- simul$xdata[-ids_train, , drop = FALSE]
y2 <- simul$ydata[-ids_train, , drop = FALSE]
ids_refit <- Resample(
  data = y2,
  tau = 0.5, family = "binomial"
)
xrefit <- x2[ids_refit, , drop = FALSE]
yrefit <- y2[ids_refit, , drop = FALSE]
xtest <- x2[-ids_refit, ]
ytest <- y2[-ids_refit, ]

# Stability selection and refitting
stab <- VariableSelection(xdata = xtrain, ydata = ytrain, family = "binomial")
refitted <- Refit(xdata = xrefit, ydata = yrefit, stability = stab)
```

```
# ROC analysis
predicted <- predict(refitted, newdata = as.data.frame(xtest))
roc <- ROC(predicted = predicted, observed = ytest)
PlotROC(roc)
plot(roc) # alternative formulation
```

---

|                   |                                  |
|-------------------|----------------------------------|
| SelectedVariables | <i>Stably selected variables</i> |
|-------------------|----------------------------------|

---

### Description

Extracts the (calibrated) set of stably selected variables.

### Usage

```
SelectedVariables(stability, argmax_id = NULL)
```

### Arguments

|           |  |
|-----------|--|
| stability | output of <a href="#">VariableSelection</a> , or <a href="#">BiSelection</a> .     |
| argmax_id | optional matrix of parameter IDs. If argmax_id=NULL, the calibrated model is used. |

### Value

A binary vector encoding the selection status of the variables (1 if selected, 0 otherwise).

### See Also

[VariableSelection](#), [BiSelection](#)

Other calibration functions: [Adjacency\(\)](#), [ArgmaxId\(\)](#), [Argmax\(\)](#), [CalibrationPlot\(\)](#), [SelectionProportions\(\)](#)

### Examples

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

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

# Calibrated set
selected <- SelectedVariables(stab)

# User-defined parameters
myids <- matrix(c(50, 10), nrow = 1)
stab$Lambda[myids[1], 1] # corresponding penalty
```

```
stab$params$pi_list[myids[2]] # corresponding threshold
selected <- SelectedVariables(stab, argmax_id = myids)
```

---

|               |                                     |
|---------------|-------------------------------------|
| SelectionAlgo | <i>Variable selection algorithm</i> |
|---------------|-------------------------------------|

---

### 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, <a href="#">LambdaGridRegression</a> 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 <a href="#">glmnet</a> . 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 = c(15, 15, 20))

# 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 <a href="#">VariableSelection</a> , <a href="#">BiSelection</a> , or <a href="#">GraphicalModel</a> . 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 <a href="#">SimulateRegression</a> , <a href="#">SimulateComponents</a> , or <a href="#">SimulateGraphical</a> . 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: [SelectionPerformanceGraph\(\)](#)

**Examples**

```

# Variable selection model
set.seed(1)
simul <- SimulateRegression(pk = 30)
stab <- VariableSelection(xdata = simul$xdata, ydata = simul$ydata)
perf <- SelectionPerformance(theta = stab, theta_star = simul)
perf <- SelectionPerformance(
  theta = SelectedVariables(stab),
  theta_star = simul$theta
) # alternative formulation

# Single-block graphical model
set.seed(1)
simul <- SimulateGraphical(pk = 30)
stab <- GraphicalModel(xdata = simul$data)
perf <- SelectionPerformance(theta = stab, theta_star = simul)
perf <- SelectionPerformance(
  theta = stab, theta_star = simul,
  cor = cor(simul$data), thr = 0.5
)
perf <- SelectionPerformance(
  theta = Adjacency(stab),
  theta_star = simul$theta
) # alternative formulation

# Multi-block graphical model
set.seed(1)
simul <- SimulateGraphical(pk = c(10, 10))
stab <- GraphicalModel(xdata = simul$data, pk = c(10, 10), lambda_other_blocks = rep(0, 3))
perf <- SelectionPerformance(theta = stab, theta_star = simul, pk = c(10, 10))
perf <- SelectionPerformance(
  theta = stab, theta_star = simul, pk = c(10, 10),
  cor = cor(simul$data), thr = 0.5
)
perf <- SelectionPerformance(
  theta = Adjacency(stab),
  theta_star = simul$theta,
  pk = c(10, 10)
) # alternative formulation

# Sparse PLS model
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),
  LambdaY = 1:(ncol(y) - 1),
  implementation = SparsePLS,

```

```

    n_cat = 2
  )
  perf <- SelectionPerformance(theta = stab, theta_star = simul)
  perf <- SelectionPerformance(
    theta = stab$selected,
    theta_star = simul$theta
  ) # alternative formulation

```

---

 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 <a href="#">GraphicalModel</a> , <a href="#">VariableSelection</a> , or <a href="#">BiSelection</a> .   |
| theta_star  | true binary adjacency matrix or output of <a href="#">SimulateGraphical</a> or <a href="#">SimulateRegression</a> .  |
| 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 <a href="#">Graph</a> .   |

**Value**

An igraph object.

**See Also**

[SimulateGraphical](#), [SimulateRegression](#), [GraphicalModel](#), [VariableSelection](#), [BiSelection](#)

Other functions for model performance: [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)

# Alternative formulation
perfgraph <- SelectionPerformanceGraph(
  theta = Adjacency(stab),
  theta_star = simul$theta
)
plot(perfgraph)

# User-defined colours/shapes
perfgraph <- SelectionPerformanceGraph(
  theta = stab, theta_star = simul,
  col = c("forestgreen", "orange", "black"),
  node_colour = "red", node_shape = "star"
)
plot(perfgraph)
perfgraph <- SelectionPerformanceGraph(
  theta = stab, theta_star = simul,
  col = c("forestgreen", "orange", "black"), lty = c(4, 2, 3)
)
plot(perfgraph)

# Using and re-formatting igraph object
require(igraph)
igraph::V(perfgraph)$size <- 10
plot(perfgraph, layout = igraph::layout_with_kk(perfgraph))

# Regression model
set.seed(1)
```

```

simul <- SimulateRegression(pk = 30)
stab <- VariableSelection(xdata = simul$xdata, ydata = simul$ydata)
perf <- SelectionPerformance(theta = stab, theta_star = simul)
perf_graph <- SelectionPerformanceGraph(theta = stab, theta_star = simul)
plot(perf_graph)

# Sparse PLS model
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),
  LambdaY = 1:(ncol(y) - 1),
  implementation = SparsePLS,
  n_cat = 2
)
perf <- SelectionPerformance(theta = stab, theta_star = simul)
perf_graph <- SelectionPerformanceGraph(theta = stab, theta_star = simul)
plot(perf_graph)

```

---

SelectionProportions    *Selection proportions*

---

### Description

Extracts the selection (or co-membership) proportions of the (calibrated) model.

### Usage

```
SelectionProportions(stability, argmax_id = NULL)
```

### Arguments

|                        |   |
|------------------------|---|
| <code>stability</code> | output of <a href="#">VariableSelection</a> , <a href="#">GraphicalModel</a> , or <a href="#">BiSelection</a> . |
| <code>argmax_id</code> | optional matrix of parameter IDs. If <code>argmax_id=NULL</code> , the calibrated model is used.                |

### Value

A symmetric matrix (graphical model) or vector (variable selection) of selection proportions.

### See Also

[VariableSelection](#), [GraphicalModel](#), [BiSelection](#)

Other calibration functions: [Adjacency\(\)](#), [ArgmaxId\(\)](#), [Argmax\(\)](#), [CalibrationPlot\(\)](#), [SelectedVariables\(\)](#)

**Examples**

```

## Variable selection

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

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

# Calibrated selection proportions
prop <- SelectionProportions(stab)

# User-defined parameters
myids <- matrix(c(80, 10), nrow = 1)
stab$Lambda[myids[1], 1] # corresponding penalty
stab$params$pi_list[myids[2]] # corresponding threshold
prop <- SelectionProportions(stab, argmax_id = myids)

## Graphical model

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

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

# Calibrated matrix of selection proportions
prop <- SelectionProportions(stab)

# User-defined parameters
myids <- matrix(c(20, 10), nrow = 1)
stab$Lambda[myids[1], 1] # corresponding penalty
stab$params$pi_list[myids[2]] # corresponding threshold
prop <- SelectionProportions(stab, argmax_id = myids)

## Dimensionality reduction

# Data simulation (continuous outcomes)
set.seed(1)
simul <- SimulateRegression(n = 50, pk = c(5, 5, 5), family = "gaussian")
x <- simul$xdata
y <- simul$ydata

# Sparse PLS
stab <- BiSelection(
  xdata = x, ydata = y,
  family = "gaussian", ncomp = 3,

```

```

    LambdaX = 1:(ncol(x) - 1),
    implementation = SparsePLS
  )

# Calibrated selection proportions per component
prop <- SelectionProportions(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 <a href="#">sgPLS</a> is run (for continuous outcomes). If family="binomial", a PLS-DA model as defined in <a href="#">sgPLSda</a> 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 <a href="#">sgPLS</a> is obtained by concatenating keepX_previous and Lambda. |
| keepY          | number of selected groups of outcome variables. This argument is defined as in <a href="#">sgPLS</a> . 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 <a href="#">sgPLS</a> or <a href="#">sgPLSda</a> .   |

**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, Thiébaud 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](https://doi.org/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 = c(10, 15, 5), 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 <a href="#">spca</a> ) or "rSVD" (for the regularised SVD approach proposed by Shen and Huang and implemented in <a href="#">spca</a> ). |
| ...            | additional arguments to be passed to <a href="#">spca</a> (if algorithm="sPCA") or <a href="#">spca</a> (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](https://doi.org/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](https://doi.org/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 <a href="#">sPLS</a> is run (for continuous outcomes). If family="binomial", a PLS-DA model as defined in <a href="#">sPLSda</a> 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 <a href="#">sPLS</a> is obtained by concatenating keepX_previous and Lambda.  |
| keepY          | number of selected outcome variables. This argument is defined as in <a href="#">sPLS</a> . Only used if family="gaussian".   |
| ...            | additional arguments to be passed to <a href="#">sPLS</a> or <a href="#">sPLSda</a> .   |

**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](https://doi.org/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 = c(10, 20, 30), 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 = 200, 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**

|                     |  |
|---------------------|--|
| <code>data</code>   | vector or matrix of data. In regression, this should be the outcome data.  |
| <code>family</code> | type of regression model. This argument is defined as in <a href="#">glmnet</a> . Possible values include "gaussian" (linear regression), "binomial" (logistic regression), "multinomial" (multinomial regression), and "cox" (survival analysis). |
| <code>tau</code>    | 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**

|                                  |   |
|----------------------------------|---|
| <code>selprop</code>             | array of selection proportions.   |
| <code>pk</code>                  | optional vector encoding the grouping structure. Only used for multi-block stability selection where <code>pk</code> indicates the number of variables in each group. If <code>pk=NULL</code> , single-block stability selection is performed.  |
| <code>pi_list</code>             | vector of thresholds in selection proportions. If <code>n_cat=3</code> , these values must be $>0.5$ and $<1$ . If <code>n_cat=2</code> , these values must be $>0$ and $<1$ .  |
| <code>K</code>                   | number of resampling iterations.  |
| <code>n_cat</code>               | number of categories used to compute the stability score. Possible values are 2 or 3.   |
| <code>PFER_method</code>         | method used to compute the upper-bound of the expected number of False Positives (or Per Family Error Rate, PFER). If <code>PFER_method="MB"</code> , the method proposed by Meinshausen and Bühlmann (2010) is used. If <code>PFER_method="SS"</code> , the method proposed by Shah and Samworth (2013) under the assumption of unimodality is used.   |
| <code>PFER_thr_blocks</code>     | vector of block-specific thresholds in PFER for constrained calibration by error control. If <code>PFER_thr=Inf</code> and <code>FDP_thr=Inf</code> , unconstrained calibration is used.  |
| <code>FDP_thr_blocks</code>      | 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 <code>PFER_thr=Inf</code> and <code>FDP_thr=Inf</code> , unconstrained calibration is used.   |
| <code>Sequential_template</code> | 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](https://doi.org/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](https://doi.org/10.1111/j.14679868.2011.01034.x).

## See Also

Other stability metric functions: [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  $\pi$ ).

The likelihood of observing stably selected, stably excluded and unstable features can be expressed as:

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

where  $H_\lambda(j)$  is the selection count of feature  $j$ .

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: `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)
```

---

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,
  ...
)

```

**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, <a href="#">LambdaGridRegression</a> 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 <a href="#">set.seed</a> ).   |
| 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 <a href="#">glmnet</a> . Possible values include "gaussian" (linear regression), "binomial" (logistic regression), "multinomial" (multinomial regression), and "cox" (survival analysis).               |

|                                 |   |
|---------------------------------|---|
| <code>implementation</code>     | function to use for variable selection. Possible functions are: <code>PenalisedRegression</code> , <code>SparsePLS</code> , <code>GroupPLS</code> and <code>SparseGroupPLS</code> . Alternatively, a user-defined function can be provided.   |
| <code>resampling</code>         | resampling approach. Possible values are: <code>"subsampling"</code> for sampling without replacement of a proportion $\tau$ of the observations, or <code>"bootstrap"</code> 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 <code>data</code> and <code>tau</code> and return the IDs of observations to be included in the resampled dataset. |
| <code>cpss</code>               | 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 <code>PFER_method="MB"</code> .   |
| <code>PFER_method</code>        | method used to compute the upper-bound of the expected number of False Positives (or Per Family Error Rate, PFER). If <code>PFER_method="MB"</code> , the method proposed by Meinshausen and Bühlmann (2010) is used. If <code>PFER_method="SS"</code> , the method proposed by Shah and Samworth (2013) under the assumption of unimodality is used.   |
| <code>PFER_thr</code>           | threshold in PFER for constrained calibration by error control. If <code>PFER_thr=Inf</code> and <code>FDP_thr=Inf</code> , unconstrained calibration is used (the default).  |
| <code>FDP_thr</code>            | threshold in the expected proportion of falsely selected features (or False Discovery Proportion) for constrained calibration by error control. If <code>PFER_thr=Inf</code> and <code>FDP_thr=Inf</code> , unconstrained calibration is used (the default).  |
| <code>Lambda_cardinal</code>    | number of values in the grid of parameters controlling the level of sparsity in the underlying algorithm. Only used if <code>Lambda=NULL</code> .   |
| <code>group_x</code>            | 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. <code>implementation=GroupPLS</code> or <code>implementation=SparseGroupPLS</code> ).  |
| <code>group_penalisation</code> | logical indicating if a group penalisation should be considered in the stability score. The use of <code>group_penalisation=TRUE</code> strictly applies to group (not sparse-group) penalisation.  |
| <code>n_cores</code>            | number of cores to use for parallel computing (see <a href="#">mclapply</a> ). Only available on Unix systems.  |
| <code>output_data</code>        | logical indicating if the input datasets <code>xdata</code> and <code>ydata</code> should be included in the output.  |
| <code>verbose</code>            | logical indicating if a loading bar and messages should be printed.   |
| <code>...</code>                | additional parameters passed to the functions provided in <code>implementation</code> or <code>resampling</code> .  |

## 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>.
- 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](https://doi.org/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](https://doi.org/10.1111/j.14679868.2011.01034.x).

## See Also

[PenalisedRegression](#), [SelectionAlgo](#), [LambdaGridRegression](#), [Resample](#), [StabilityScoreRefit](#), [ExplanatoryPerformance](#), [PlotROC](#), [Incremental](#), [PlotIncremental](#)

Other stability selection functions: [BiSelection\(\)](#), [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)

# 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 = c(20, 30), family = "gaussian")
stab <- VariableSelection(xdata = simul$xdata, ydata = simul$ydata, family = "mgaussian")
summary(stab)

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

# Multinomial regression
set.seed(1)
simul <- SimulateRegression(n = 200, pk = 15, family = "multinomial")
stab <- VariableSelection(
  xdata = simul$xdata, ydata = simul$ydata,
  family = "multinomial"
)
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)
```

# Index

- \* **calibration functions**
    - Adjacency, [5](#)
    - Argmax, [7](#)
    - ArgmaxId, [8](#)
    - CalibrationPlot, [18](#)
    - SelectedVariables, [78](#)
    - SelectionProportions, [85](#)
  - \* **ensemble model functions**
    - Ensemble, [23](#)
    - EnsemblePredictions, [24](#)
  - \* **functions for model performance**
    - SelectionPerformance, [80](#)
    - SelectionPerformanceGraph, [83](#)
  - \* **lambda grid functions**
    - LambdaGridGraphical, [51](#)
    - LambdaGridRegression, [54](#)
    - LambdaSequence, [56](#)
  - \* **multi-block functions**
    - BlockLambdaGrid, [17](#)
  - \* **penalised dimensionality reduction functions**
    - GroupPLS, [44](#)
    - SparseGroupPLS, [87](#)
    - SparsePCA, [89](#)
    - SparsePLS, [91](#)
  - \* **prediction performance functions**
    - ExplanatoryPerformance, [25](#)
    - Incremental, [47](#)
    - PlotIncremental, [61](#)
    - PlotROC, [64](#)
    - ROC, [76](#)
  - \* **stability metric functions**
    - FDP, [31](#)
    - PFER, [60](#)
    - StabilityMetrics, [94](#)
    - StabilityScore, [98](#)
  - \* **stability selection functions**
    - BiSelection, [9](#)
    - GraphicalModel, [38](#)
    - VariableSelection, [99](#)
  - \* **underlying algorithm functions**
    - PenalisedGraphical, [57](#)
    - PenalisedRegression, [58](#)
  - \* **wrapping functions**
    - GraphicalAlgo, [36](#)
    - SelectionAlgo, [79](#)
- Adjacency, [5](#), [8](#), [9](#), [20](#), [34](#), [42](#), [78](#), [85](#)
- AggregatedEffects, [6](#)
- Argmax, [5](#), [7](#), [9](#), [20](#), [78](#), [85](#)
- ArgmaxId, [5](#), [8](#), [8](#), [20](#), [78](#), [85](#)
- BiSelection, [6](#), [8](#), [9](#), [19](#), [20](#), [35](#), [42](#), [46](#), [62](#), [65](#), [67](#), [70](#), [78](#), [81](#), [83–85](#), [88](#), [90](#), [92](#), [103](#)
- BlockLambdaGrid, [17](#)
- BlockStructure, [81](#)
- CalibrationPlot, [5](#), [8](#), [9](#), [12](#), [18](#), [41](#), [78](#), [85](#), [102](#)
- coef, [27](#)
- Combine, [22](#), [41](#), [102](#)
- concordance, [27](#), [48](#)
- coxph, [26](#), [47](#), [70](#), [71](#)
- Ensemble, [23](#), [24](#)
- EnsemblePredictions, [23](#), [24](#)
- ExplanatoryPerformance, [25](#), [47](#), [49](#), [62](#), [64](#), [65](#), [77](#), [103](#)
- FDP, [31](#), [60](#), [94](#), [97](#), [99](#)
- Folds, [31](#)
- glassoFast, [37](#), [39](#), [52](#), [53](#), [57](#), [58](#)
- glm, [26](#), [47](#), [70](#), [71](#)
- glmnet, [32](#), [55](#), [58](#), [59](#), [75](#), [79](#), [93](#), [100](#)
- gPLS, [45](#)
- gPLSda, [45](#)
- Graph, [32](#), [36](#), [41](#), [42](#), [83](#)
- graph\_from\_adjacency\_matrix, [33](#)

- GraphComparison, 35
- GraphicalAlgo, 36, 42, 80
- GraphicalModel, 5, 8, 9, 14, 18–20, 22, 33–35, 37, 38, 58, 81, 83–85, 103
- GroupPLS, 14, 44, 80, 88, 90, 92
  
- igraph, 32, 33, 35, 41
- Incremental, 27, 47, 61, 62, 65, 77, 103
  
- LambdaGridGraphical, 37, 39, 42, 51, 56
- LambdaGridRegression, 53, 54, 56, 79, 100, 103
- LambdaSequence, 53, 56, 56
- lm, 26, 47, 70, 71
  
- mclapply, 11, 12, 40, 41, 101, 102
- mean, 6
- median, 6
- multinom, 26, 47, 70, 71
  
- par, 20, 62, 65
- PenalisedGraphical, 37, 42, 57, 59
- PenalisedRegression, 58, 58, 80, 103
- PFER, 31, 60, 94, 97, 99
- plotCI, 62
- PlotIncremental, 27, 49, 61, 65, 77, 103
- PlotROC, 27, 49, 62, 64, 77, 103
- PLS, 66, 69, 70
- pls, 66, 67
- points, 19, 61
- predict, 24
- predict.pls, 69
- PredictPLS, 69
  
- RCy3, 33
- Recalibrate (Refit), 70
- Refit, 6, 27, 49, 62, 65, 70, 103
- Resample, 14, 42, 75, 103
- ROC, 27, 49, 62, 65, 76
  
- SelectedVariables, 5, 8, 9, 20, 78, 85
- SelectionAlgo, 37, 59, 79, 103
- SelectionPerformance, 80, 84
- SelectionPerformanceGraph, 36, 81, 83
- SelectionProportions, 5, 8, 9, 20, 78, 85
- set.seed, 11, 39, 55, 100
- sgPLS, 45, 87, 88
- sgPLSda, 87, 88
- sharp-package, 3
- SimulateComponents, 81
- SimulateGraphical, 35, 81, 83, 84
- SimulateRegression, 35, 81, 83, 84
- SparseGroupPLS, 14, 46, 80, 87, 90, 92
- SparsePCA, 14, 46, 80, 88, 89, 92
- SparsePLS, 14, 46, 80, 88, 90, 91
- spca, 89, 90
- Split, 93
- sPLS, 91
- sPLSda, 91
- Square, 94
- StabilityMetrics, 31, 60, 94, 99
- StabilityScore, 12, 14, 31, 41, 42, 60, 94, 97, 98, 102, 103
  
- text, 20
  
- VariableSelection, 6, 8, 9, 14, 19, 20, 22–24, 26, 27, 35, 42, 46, 47, 49, 59, 62, 65, 67, 70, 71, 78, 80, 81, 83–85, 88, 90, 92, 99
- visNetwork, 33, 41