Package: mglasso (via r-universe)

September 11, 2024

Type Package

Title Multiscale Graphical Lasso

Version 0.1.3

Description Inference of Multiscale graphical models with neighborhood selection approach. The method is based on solving a convex optimization problem combining a Lasso and fused-group Lasso penalties. This allows to infer simultaneously a conditional independence graph and a clustering partition. The optimization is based on the Continuation with Nesterov smoothing in a Shrinkage-Thresholding Algorithm solver (Hadj-Selem et al. 2018) <doi:10.1109/TMI.2018.2829802> implemented in python.

License MIT + file LICENSE

Imports corpcor, ggplot2, ggrepel, gridExtra, Matrix, methods, R.utils, reticulate (>= 1.25), rstudioapi, capushe, DescTools

Suggests knitr, mvtnorm, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

ByteCompile true

Config/reticulate list(packages = list(list(package = ``scipy", version = ``1.7.1"), list(package = ``numpy", version = ``1.22.4"), list(package = ``matplotlib"), list(package = ``scikit-learn"), list(package = ``six"), list(package = ``pylearn-parsimony", version = ``0.3.1", pip = TRUE)))

Encoding UTF-8

RoxygenNote 7.2.3

URL https://desanou.github.io/mglasso/

Roxygen list(markdown = TRUE)

Config/testthat/edition 3

Repository https://desanou.r-universe.dev

RemoteUrl https://github.com/desanou/mglasso

RemoteRef HEAD

RemoteSha 0cbbafe519d84dcc75b0461e8b92e13720cc64ce

Contents

`adj_mat` *Adjacency matrix*

Description

Adjacency matrix

Usage

```
adj_mat(mat, sym_rule = "and")
```

Arguments

`mat`	matrix of regression coefficients
`sym_rule`	symmetrization rule, either AND or OR

Value

adjacency matrix

`beta_idty` *Init Beta 1 matrix*

Description

Init Beta 1 matrix

Init Beta 1 matrix

Usage

```
beta_idty(p)
```

```
beta_idty(p)
```

beta_ols *Init Beta via OLS*

Description

Init Beta via OLS

Init Beta via OLS

Initialize regression matrix

Usage

```
beta_ols(X)

beta_ols(X)

beta_ols(X)
```

Arguments

X	data

Value

A zero-diagonal matrix of regression vectors.

beta_to_vector *vectorize beta matrix*

Description

vectorize beta matrix

vectorize beta matrix

Transform a matrix of regression coefficients to vector removing the diagonal

Usage

```
beta_to_vector(beta_mat)

beta_to_vector(beta_mat)

beta_to_vector(beta_mat)
```

Arguments

beta_mat	matrix of regressions vectors

Value

A numeric vector of all regression coefficients.

`bloc_diag` *return precision matrix*

Description

return precision matrix

Usage

```
bloc_diag(n_vars, connectivity_mat, prop_clusters, rho)
```

`cah_glasso` *cah_glasso*

Description

cah_glasso

Usage

```
cah_glasso(num_clusters, data, lam1, hclust_obj)
```

`conesta` *CONESTA solver.*

Description

Solve the MGLasso optimization problem using CONESTA algorithm. Interface to the pylearn.parsimony python library.

Usage

```
conesta(
  X,
  lam1,
  lam2,
  beta_warm = c(0),
  type_ = "initial",
  W_ = NULL,
  mean_ = FALSE,
  max_iter_ = 10000,
  prec_ = 0.01
)
```

Arguments

X	Data matrix nxp.
lam1	Sparsity penalty.
lam2	Total variation penalty.
beta_warm	Warm initialization vector.
type_	Character scalar. By default set to initial version which doesn't use weights
W_	Weights matrix for total variation penalties.
mean_	Logical scalar. If TRUE weights the optimization function by the inverse of sample size.
max_iter_	Numeric scalar. Maximum number of iterations.
prec_	Numeric scalar. Tolerance for the stopping criterion (duality gap).

Details

COntinuation with NEsterov smoothing in a Shrinkage-Thresholding Algorithm (CONESTA, Hadj-Selem et al. 2018) doi:10.1109/TMI.2018.2829802 is an algorithm design for solving optimization problems including group-wise penalties. This function is an interface with the python solver. The MGLasso problem is first reformulated in a problem of the form

$$argmin 1/2||Y - \tilde{X}\tilde{\beta}||_2^2 + \lambda_1||\tilde{\beta}||_1 + \lambda_2 \sum_{i<j} ||\boldsymbol{A}_{ij}\tilde{\beta}||_2$$

where vector Y is the vectorized form of matrix X.

Value

Numeric matrix of size pxp. Line k of the matrix represents the coefficients obtained from the L1-L2 penalized regression of variable k on the others.

See Also

mglasso() for the MGLasso model estimation.

Examples

```
## Not run: # because of installation of external packages during checks
mglasso::install_pylearn_parsimony(envname = "rmglasso", method = "conda")
reticulate::use_condaenv("rmglasso", required = TRUE)
reticulate::py_config()
n = 30
K = 2
p = 4
rho = 0.85
blocs <- list()
for (j in 1:K) {
 bloc <- matrix(rho, nrow = p/K, ncol = p/K)
   for(i in 1:(p/K)) { bloc[i,i] <- 1 }
   blocs[[j]] <- bloc
```

```
   }

mat.covariance <- Matrix::bdiag(blocs)
mat.covariance
set.seed(11)
X <- mvtnorm::rmvnorm(n, mean = rep(0,p), sigma = as.matrix(mat.covariance))
X <- scale(X)
res <- conesta(X, 0.1, 0.1)

## End(Not run)
```

---

conesta_rwrapper *CONESTA solver for numerical experiments.*

Description

CONESTA solver for numerical experiments.

Usage

```
conesta_rwrapper(
  X,
  lam1,
  lam2,
  beta_warm = c(0),
  type_ = "initial",
  W_ = NULL,
  mean_ = FALSE,
  max_iter_ = 10000,
  prec_ = 0.01
)
```

Examples

```
## Not run: # because of installation of external packages during checks
mglasso::install_pylearn_parsimony(envname = "rmglasso", method = "conda")
reticulate::use_condaenv("rmglasso", required = TRUE)
reticulate::py_config()
n = 30
K = 2
p = 4
rho = 0.85
blocs <- list()
for (j in 1:K) {
 bloc <- matrix(rho, nrow = p/K, ncol = p/K)
   for(i in 1:(p/K)) { bloc[i,i] <- 1 }
   blocs[[j]] <- bloc
   }
```

```
mat.covariance <- Matrix::bdiag(blocs)
mat.covariance
set.seed(11)
X <- mvtnorm::rmvnorm(n, mean = rep(0,p), sigma = as.matrix(mat.covariance))
X <- scale(X)
res <- conesta_rwrapper(X, 0.1, 0.1)

## End(Not run)
```

`cost` *cost function*

Description

`cost` computes the cost function of `Mglasso` method.

Usage

```
cost(beta, x, lambda1 = 0, lambda2 = 0)

cost(beta, x, lambda1 = 0, lambda2 = 0)

cost(beta, x, lambda1 = 0, lambda2 = 0)
```

Arguments

`beta`	p by p numeric matrix. In rows, regression vectors coefficients after node-wise regression. `diag(beta)` = `0`.
`x`	n by p numeric matrix. Data with variables in columns.
`lambda1`	numeric scalar. Lasso penalization parameter.
`lambda2`	numeric scalar. Fused-group Lasso penalization parameter.

Value

numeric scalar. The cost.

dist_beta *distances Beta*

Description

distances Beta

Compute distance matrix between regression vectors

Usage

```
dist_beta(beta, distance = "euclidean")

dist_beta(beta, distance = "euclidean")
```

Arguments

beta	matrix of regression vectors
distance	euclidean or relative distance

Value

A numeric matrix of distances.

error *Mean error from classical regression*

Description

Mean error from classical regression

Usage

```
error(Theta, X)
```

error_huge *Formula from Huge paper*

Description

Formula from Huge paper

Usage

```
error_huge(Theta, X)
```

`expand_beta` *TO DO: Fill upper triangular matrix then sum up with the transpose to have full matrix*

Description

TO DO: Fill upper triangular matrix then sum up with the transpose to have full matrix

Usage

```
expand_beta(beta_level, clusters)
```

`expand_beta_deprecated` *doesn't work when dealing with matrix where diagonal of zero should be adjusted*

Description

doesn't work when dealing with matrix where diagonal of zero should be adjusted

Usage

```
expand_beta_deprecated(beta_level, clusters)
```

`extract_meta` *extracts meta-variables indices*

Description

extracts meta-variables indices

Usage

```
extract_meta(full_graph = NULL, clusters)
```

fun_lines *weighted sum/difference of two regression vectors*

Description

fun_lines applies function fun to regression vectors while reordering the coefficients, such that the j-th coefficient in beta[j,] is permuted with the i-th coefficient.

Usage

```
fun_lines(i, j, beta, fun = `-`, ni = 1, nj = 1)
```

Arguments

i	integer scalar. Index of the first vector.
j	integer scalar. Index of the second vector.
beta	p by p numeric matrix. In rows, regression vectors coefficients after node-wise regression. diag(beta) = 0.
fun	function. Applied on lines.
ni	integer scalar. Weight for vector i.
nj	integer scalar. Weight for vector j.

Value

numeric vector

Examples

```
beta <- matrix(round(rnorm(9),2), ncol = 3)
diag(beta) <- 0
beta
fun_lines(1, 2, beta)
fun_lines(2, 1, beta)
```

get_auc *Plot ROC curve and calculate AUC*

Description

Plot ROC curve and calculate AUC

Usage

```
get_auc(omega_hat_list, omega, to = to_)
```

Arguments

type	Classical ROC curve tpr = f(FPR) or TPR = f(precision) adjusted version. Compute AUC and partial AUC

---

get_range_nclusters *compute range of number of clusters from ROC outputs take in parameter an object from reorder_mglasso_roc_calculations*

Description

compute range of number of clusters from ROC outputs take in parameter an object from reorder_mglasso_roc_calculations

Usage

```
get_range_nclusters(out, thresh_fuse = 1e-06, p = 40)
```

---

ggplot_roc *Title*

Description

Title

Usage

```
ggplot_roc(
  omega_hat_list,
  omega,
  type = c("classical", "precision_recall"),
  main = NULL
)
```

Arguments

main

---

graph_estimate *Neighborhood selection estimate*

Description

Neighborhood selection estimate

Usage

```
graph_estimate(rho, X)
```

image_sparse *Plot the image of a matrix*

Description

Plot the image of a matrix

Usage

```
image_sparse(matrix, main_ = "", sub_ = "", col_names = FALSE)
```

Arguments

matrix	matrix of regression coefficients
main_	title
sub_	subtitle
col_names	columns names

Value

No return value.

install_pylearn_parsimony *Install the python library pylearn-parsimony and other required libraries*

Description

pylearn-parsimony contains the solver CONESTA used for the mglasso problem and is available on github at https://github.com/neurospin/pylearn-parsimony It is advised to use a python version ">=3.7,<3.10". Indeed, the latest version of scipy under which mglasso was developped is scipy 1.7.1 which is based on python ">=3.7,<3.10". In turn, this version of scipy can only be associated with a version of numpy ">=1.16.5,<1.23.0"

Usage

```
install_pylearn_parsimony(
  method = c("auto", "virtualenv", "conda"),
  conda = "auto",
  extra_pack = c("scipy == 1.7.1", "scikit-learn", "numpy == 1.22.4", "six",
    "matplotlib"),
  python_version = "3.8",
  restart_session = TRUE,
  envname = NULL,
  ...
)
```

Arguments

method	Installation method. By default, "auto" automatically finds a method that will work in the local environment. Change the default to force a specific installation method. Note that the "virtualenv" method is not available on Windows.
conda	The path to a conda executable. Use "auto" to allow reticulate to automatically find an appropriate conda binary. See **Finding Conda** and conda_binary() for more details.
extra_pack	Character vector. Extra-packages to be installed.
python_version	The requested Python version. Ignored when attempting to install with a Python virtual environment.
restart_session	Restart R session after installing (note this will only occur within RStudio)
envname	The name, or full path, of the environment in which Python packages are to be installed. When NULL (the default), the active environment as set by the RETICULATE_PYTHON_ENV variable will be used; if that is unset, then the r-reticulate environment will be used.
...	additionnal arguments passed to reticulate::py_install()

Value

No return value.

---

lagrangian *lagrangian function*

Description

Beta and X must have the same number of variables

Beta and X must have the same number of variables

Usage

```
lagrangian(Beta, X, lambda1 = 0, lambda2 = 0)

lagrangian(Beta, X, lambda1 = 0, lambda2 = 0)
```

Arguments

Beta	numeric matrix. In rows, regression vectors coefficients following of node-wise regression. diag(Beta) = 0
X	numeric matrix. Data with variables in columns.
lambda1	numeric scalar. Lasso penalization parameter.
lambda2	numeric scalar. Fused-group Lasso penalization parameter.

Value

numeric scalar. The lagrangian

numeric scalar. The lagrangian

Examples

```
## Generation of K block partitions
n = 50
K = 3
p = 6
rho = 0.85
blocs <- list()
for (j in 1:K) {
   bloc <- matrix(rho, nrow = p/K, ncol = p/K)
   for(i in 1:(p/K)) { bloc[i,i] <- 1 }
   blocs[[j]] <- bloc
   }
mat.covariance <- bdiag(blocs)
set.seed(11)
X <- rmvnorm(n, mean = rep(0,p), sigma = as.matrix(mat.covariance))
X <- scale(X)

## Initialization for Beta
Beta1 <- matrix(0, nrow = p, ncol = p)
for(i in 1:p){
  Beta1[i,-i] <- solve(t(X[,-i])%*%X[,-i]) %*% t(X[,-i]) %*% X[,i]
  }
lagrangian(Beta, X, 0, 0)

## Generation of K block partitions
n = 50
K = 3
p = 6
rho = 0.85
blocs <- list()
for (j in 1:K) {
   bloc <- matrix(rho, nrow = p/K, ncol = p/K)
   for(i in 1:(p/K)) { bloc[i,i] <- 1 }
   blocs[[j]] <- bloc
   }
mat.covariance <- bdiag(blocs)
set.seed(11)
X <- rmvnorm(n, mean = rep(0,p), sigma = as.matrix(mat.covariance))
X <- scale(X)

## Initialization for Beta
Beta1 <- matrix(0, nrow = p, ncol = p)
for(i in 1:p){
  Beta1[i,-i] <- solve(t(X[,-i])%*%X[,-i]) %*% t(X[,-i]) %*% X[,i]
  }
lagrangian(Beta, X, 0, 0)
```

lasso_estimate *Check first estimate coeffs with glm*

Description

Check first estimate coeffs with glm

Usage

```
lasso_estimate(response_variable_number, penalty_value)
```

mean_prec_mat *mean of randomly simulated precision matrices in the same configuration*

Description

mean of randomly simulated precision matrices in the same configuration

Usage

```
mean_prec_mat(nrep = 10, config = config_)
```

mergeX *Merge X*

Description

weighted mean

Usage

```
mergeX(X, pair_to_merge, clusters)
```

merge_beta *Merge Beta Different types of merging and their effect*

Description

Merge Beta Different types of merging and their effect

Usage

```
merge_beta(Beta, pair_to_merge, clusters)
```

merge_clusters *compute clusters partition from pairs of variables to merge*

Description

compute clusters partition from pairs of variables to merge

Usage

```
merge_clusters(pairs_to_merge, clusters)
```

Arguments

pairs_to_merge	table of the indices of variables to be merge
clusters	numeric vector. By default 1:p where p is the number of variables

Value

A numeric vector.

merge_labels *Merge labels*

Description

Merge labels

Usage

```
merge_labels(merged_pair, labels, level)
```

merge_proc *merge clusters from table*

Description

merge clusters from table

Usage

```
merge_proc(
  pairs_to_merge,
  clusters,
  X,
  Beta,
  level,
  gain_level,
  gains,
  labels,
  merge
)
```

mglasso *Inference of Multiscale Gaussian Graphical Model.*

Description

Cluster variables using L2 fusion penalty and simultaneously estimates a gaussian graphical model structure with the addition of L1 sparsity penalty.

Usage

```
mglasso(
  x,
  lambda1 = 0,
  fuse_thresh = 0.001,
  maxit = NULL,
  distance = c("euclidean", "relative"),
  lambda2_start = 1e-04,
  lambda2_factor = 1.5,
  precision = 0.01,
  weights_ = NULL,
  type = c("initial"),
  compact = TRUE,
  verbose = FALSE
)
```

Arguments

x	Numeric matrix (nxp). Multivariate normal sample with n independent observations.
lambda1	Positive numeric scalar. Lasso penalty.
fuse_thresh	Positive numeric scalar. Threshold for clusters fusion.
maxit	Integer scalar. Maximum number of iterations.
distance	Character. Distance between regression vectors with permutation on symmetric coefficients.
lambda2_start	Numeric scalar. Starting value for fused-group Lasso penalty (clustering penalty).
lambda2_factor	Numeric scalar. Step used to update fused-group Lasso penalty in a multiplicative way..
precision	Tolerance for the stopping criterion (duality gap).
weights_	Matrix of weights.
type	If "initial" use classical version of **MGLasso** without weights.
compact	Logical scalar. If TRUE, only save results when previous clusters are different from current.
verbose	Logical scalar. Print trace. Default value is FALSE.

Details

Estimates a gaussian graphical model structure while hierarchically grouping variables by optimizing a pseudo-likelihood function combining Lasso and fuse-group Lasso penalties. The problem is solved via the *COntinuation with NEsterov smoothing in a Shrinkage-Thresholding Algorithm* (Hadj-Selem et al. 2018). Varying the fusion penalty λ_2 in a multiplicative fashion allow to uncover a seemingly hierarchical clustering structure. For $\lambda_2 = 0$, the approach is theoretically equivalent to the Meinshausen-Bühlmann (2006) *neighborhood selection* as resuming to the optimization of *pseudo-likelihood* function with ℓ_1 penalty (Rocha et al., 2008). The algorithm stops when all the variables have merged into one cluster. The criterion used to merge clusters is the ℓ_2-norm distance between regression vectors.

For each iteration of the algorithm, the following function is optimized :

$$1/2 \sum_{i=1}^{p} ||X^i - X^{\ i}\beta^i||_2^2 + \lambda_1 \sum_{i=1}^{p} ||\beta^i||_1 + \lambda_2 \sum_{i<j} ||\beta^i - \tau_{ij}(\beta^j)||_2.$$

where β^i is the vector of coefficients obtained after regression X^i on the others and τ_{ij} is a permutation exchanging β^i_j with β^j_i.

Value

A list-like object of class `mglasso` is returned.

out	List of lists. Each element of the list corresponds to a clustering level. An element named `levelk` contains the regression matrix `beta` and clusters vector `clusters` for a clustering in k clusters. When `compact = TRUE` `out` has as many elements as the number of unique partitions. When set to `FALSE`, the function returns as many items as the the range of values taken by `lambda2`.
l1	the sparsity penalty `lambda1` used in the problem solving.

See Also

conesta() for the problem solver, plot_mglasso() for plotting the output of mglasso.

Examples

```
## Not run:
mglasso::install_pylearn_parsimony(envname = "rmglasso", method = "conda")
reticulate::use_condaenv("rmglasso", required = TRUE)
reticulate::py_config()
n = 50
K = 3
p = 9
rho = 0.85
blocs <- list()
for (j in 1:K) {
  bloc <- matrix(rho, nrow = p/K, ncol = p/K)
  for(i in 1:(p/K)) { bloc[i,i] <- 1 }
  blocs[[j]] <- bloc
}

mat.covariance <- Matrix::bdiag(blocs)
mat.covariance

set.seed(11)
X <- mvtnorm::rmvnorm(n, mean = rep(0,p), sigma = as.matrix(mat.covariance))
X <- scale(X)

res <- mglasso(X, 0.1, lambda2_start = 0.1)
res$out[[1]]$clusters
res$out[[1]]$beta

## End(Not run)
```

---

neighbor_select *neighbor_select*

Description

neighbor_select

Usage

```
neighbor_select(
  data = data$X,
  config,
  lambda_min_ratio = 0.01,
  nlambda = 10,
  nresamples = 20,
  lambdas = NULL,
```

```
  model = NULL,
  verbose = FALSE,
  estim_var = NULL
)
```

`one_config` *One simulation configuration*

Description

One simulation configuration

Usage

```
one_config(n, p, pi, alpha, rho)
```

`perf_one` *compute TPR, FPR, SHD given estimated and true precision matrices*

Description

compute TPR, FPR, SHD given estimated and true precision matrices

Usage

```
perf_one(omega_hat, omega)
```

Details

SHD: structural hamming distance

`perf_vec` *get performances from list of estimations*

Description

get performances from list of estimations

Usage

```
perf_vec(omega_hat_list, omega)
```

plot_clusterpath *Plot MGLasso Clusterpath*

Description

Plot MGLasso Clusterpath

Usage

```
plot_clusterpath(X, mglasso_res, colnames_ = NULL)
```

Arguments

X	numeric matrix
mglasso_res	object of class mglasso
colnames_	columns labels

Details

This function plot the clustering path of mglasso method on the 2 principal components axis of X. As the centroids matrices are not of the same dimension as X, we choose to plot the predicted X matrix path.

Value

no return value.

plot_mglasso *fonction qui affiche les matrices d'adjacence à chaque niveau de la hiérarchie à automatiser utiliser niveau de legende commune*

Description

Plot the object returned by the mglasso function.

Usage

```
plot_mglasso(mglasso_, levels_ = NULL)

plot_mglasso(mglasso_, levels_ = NULL)
```

Arguments

mglasso_	Object of class mglasso.
levels_	Character vector. Selected levels for which estimated matrices will be plot. If NULL plot all levels.

Value

No return value.

`precision_to_regression` *Compute precision matrix from regression vectors*

Description

Compute precision matrix from regression vectors

Usage

```
precision_to_regression(K)
```

Arguments

`K`	precision matrix

Value

A numeric matrix.

`repart` *pareil pour les clusters ACP dimensions ?*

Description

pareil pour les clusters ACP dimensions ?

Usage

```
repart(cor_)
```

`select_ebic_weighted` *EBIC*

Description

EBIC

Usage

```
select_ebic_weighted(Thetas, ploglik, n_edges, n, p, gam = 0.5, pen_params)
```

select_kfold *K-fold cross validation neghborhood lasso selection*

Description

K-fold cross validation neghborhood lasso selection

Usage

```
select_kfold(
  X,
  Thetas,
  lambdas = NULL,
  n_lambda,
  K_fold = 10,
  criterion = "ploglik",
  verbose = TRUE
)
```

Arguments

criterion	c("ploglik", "rmse")

Details

if criterion = "ploglik" use pseudo-log likelihood formula from Huge paper with matrix approach if "rmse" use mean squarred prediction error

select_kfold_mglasso *K-fold cross validation mglasso*

Description

K-fold cross validation mglasso

Usage

```
select_kfold_mglasso(
  X,
  lambda1s = NULL,
  lambda2s = NULL,
  K_fold = 5,
  nl1 = 1,
  nl2 = 1,
  lam1_min_ratio,
  verbose = TRUE
)
```

select_partition *Finds the optimal number of clusters using slope heuristic*

Description

Finds the optimal number of clusters using slope heuristic

Usage

```
select_partition(gains)
```

Value

integer scalar. The indice of the selected model.

select_stab_mglasso *stability selection mglasso*

Description

stability selection mglasso

Usage

```
select_stab_mglasso(X, l1_, l2_, subsample_ratio, nrep, stab_thresh)
```

select_stars_mglasso *stability selection mglasso II stars way*

Description

stability selection mglasso II stars way

Usage

```
select_stars_mglasso(
  X,
  lambda1s = NULL,
  lambda2s = NULL,
  subsample_ratio = NULL,
  nrep = 1,
  stars_thresh = 0.1,
  nl1 = 1,
  nl2 = 1
)
```

seq_l1l2 *def sequences for lambda1s and lambda2s not sure if max of lambda1 s still the same as in the lasso case. But if find better equivalence will update this part*

Description

def sequences for lambda1s and lambda2s not sure if max of lambda1 s still the same as in the lasso case. But if find better equivalence will update this part

Usage

```
seq_l1l2(
  X,
  nlam1 = 2,
  nlam2 = 2,
  logscale = TRUE,
  mean = FALSE,
  lambda1_min_ratio = 0.01,
  require_non_list = FALSE,
  l2_max = NULL
)
```

Arguments

mean	in conesta_rwrapper is the mean criterion used ie averaged by np

sim_data *simulate data with given graph structure*

Description

simulate data with given graph structure

Usage

```
sim_data(
  p = 20,
  np_ratio = 2,
  structure = c("block_diagonal", "hub", "scale_free", "erdos"),
  alpha,
  prob_mat,
  rho,
  g,
  inter_cluster_edge_prob = 0.01,
```

```
  p_erdos = 0.1,
  verbose = FALSE
)
```

Arguments

verbose

Value

A list: graph : precision

---

symmetrize *symmetrize matrix of regression vectors pxp*

Description

symmetrize matrix of regression vectors pxp

Apply symmetrization on estimated graph

Usage

```
symmetrize(mat, rule = "and")

symmetrize(mat, rule = "and")
```

Arguments

mat	graph or precision matrix
rule	"and" or "or" rule

Value

A numeric matrix.

Index