Package ‘spectralGraphTopology’

October 14, 2022

Title  Learning Graphs from Data via Spectral Constraints
Version  0.2.3
Date  2022-03-12

Description  In the era of big data and hyperconnectivity, learning high-dimensional structures such as graphs from data has become a prominent task in machine learning and has found applications in many fields such as finance, health care, and networks. ‘spectralGraphTopology’ is an open source, documented, and well-tested R package for learning graphs from data. It provides implementations of state of the art algorithms such as Combinatorial Graph Laplacian Learning (CGL), Spectral Graph Learning (SGL), Graph Estimation based on Majorization-Minimization (GLE-MM), and Graph Estimation based on Alternating Direction Method of Multipliers (GLE-ADMM). In addition, graph learning has been widely employed for clustering, where specific algorithms are available in the literature. To this end, we provide an implementation of the Constrained Laplacian Rank (CLR) algorithm.

Maintainer  Ze Vinicius <jvmirca@gmail.com>


BugReports  https://github.com/dppalomar/spectralGraphTopology/issues

Depends
License  GPL-3
Encoding  UTF-8
LinkingTo  Rcpp, RcppArmadillo, RcppEigen
Imports  Rcpp (>= 0.11.0), MASS, Matrix, progress, rlist
RoxygenNote  7.1.1
Suggests  CVXR, bookdown, knitr, prettydoc, rmarkdown, R.rsp, testthat, patrick, corrplot, igraph, kernlab, pals, clusterSim, viridis, quadprog, matrixcalc
VignetteBuilder  CVXR, knitr, rmarkdown, R.rsp
Description

This package provides estimators to learn k-component, bipartite, and k-component bipartite graphs from data by imposing spectral constraints on the eigenvalues and eigenvectors of the Laplacian and adjacency matrices. Those estimators leverage spectral properties of the graphical models as a prior information, which turn out to play key roles in unsupervised machine learning tasks such as community detection.
Functions

- learn_k_component_graph
- learn_bipartite_graph
- learn_bipartite_k_component_graph
- cluster_k_component_graph
- learn_laplacian_gle_mm
- learn_laplacian_gle_admm

Help

For a quick help see the README file: GitHub-README.

Author(s)

Ze Vinicius and Daniel P. Palomar

References


\[
A \ 
\text{Computes the Adjacency linear operator which maps a vector of weights into a valid Adjacency matrix.}
\]

Description

Computes the Adjacency linear operator which maps a vector of weights into a valid Adjacency matrix.

Usage

\[ A(w) \]

Arguments

- \( w \) weight vector of the graph

Value

\( Aw \) the Adjacency matrix

Examples

\[
\text{library(spectralGraphTopology)}
\text{A_w <- A(c(1, 0, 1))}
\text{A_w}
\]
**accuracy**  
*Computes the accuracy between two matrices*

**Description**  
Computes the accuracy between two matrices

**Usage**  
```
accuracy(Wtrue, West, eps = 1e-04)
```

**Arguments**  
- `Wtrue`: true matrix  
- `West`: estimated matrix  
- `eps`: real number such that edges whose values are smaller than eps are not considered in the computation of the fscore

**Examples**  
```
library(spectralGraphTopology)
X <- L(c(1, 0, 1))
accuracy(X, X)
```

**Astar**  
*Computes the Astar operator.*

**Description**  
Computes the Astar operator.

**Usage**  
```
Astar(M)
```

**Arguments**  
- `M`: matrix

**Value**  
- `w` vector
**block_diag**

*Constructs a block diagonal matrix from a list of square matrices*

**Description**

Constructs a block diagonal matrix from a list of square matrices

**Usage**

`block_diag(...)`

**Arguments**

... list of matrices or individual matrices

**Value**

block diagonal matrix

**Examples**

```r
library(spectralGraphTopology)
X <- L(c(1, 0, 1))
Y <- L(c(1, 0, 1, 0, 0, 1))
B <- block_diag(X, Y)
B
```

---

**cluster_k_component_graph**

*Cluster a k-component graph from data using the Constrained Laplacian Rank algorithm*

Cluster a k-component graph on the basis of an observed data matrix. Check out https://mirca.github.io/spectralGraphTopology for code examples.

**Description**

Cluster a k-component graph from data using the Constrained Laplacian Rank algorithm

Cluster a k-component graph on the basis of an observed data matrix. Check out https://mirca.github.io/spectralGraphTopology for code examples.
cluster_k_component_graph

Usage

cluster_k_component_graph(
    Y,
    k = 1,
    m = 5,
    lmd = 1,
    eigtol = 1e-09,
    edgetol = 1e-06,
    maxiter = 1000
)

Arguments

Y          a pxn data matrix, where p is the number of nodes and n is the number of features (or data points per node)

k          the number of components of the graph

m          the maximum number of possible connections for a given node used to build an affinity matrix

lmd        L2-norm regularization hyperparameter

eigtol     value below which eigenvalues are considered to be zero

edgetol    value below which edge weights are considered to be zero

maxiter    the maximum number of iterations

Value

A list containing the following elements:

laplacian  the estimated Laplacian Matrix

adjacency  the estimated Adjacency Matrix

eigvals    the eigenvalues of the Laplacian Matrix

lmd_seq    sequence of lmd values at every iteration

elapsed_time elapsed time at every iteration

Author(s)

Ze Vinicius and Daniel Palomar

References

Examples

```r
library(clusterSim)
library(spectralGraphTopology)
library(igraph)
set.seed(1)
# number of nodes per cluster
N <- 30
# generate datapoints
twomoon <- shapes.two.moon(N)
# estimate underlying graph
graph <- cluster_k_component_graph(twomoon$data, k = 2)
# build network
net <- graph_from_adjacency_matrix(graph$adjacency, mode = "undirected", weighted = TRUE)
# colorify nodes and edges
colors <- c("#706FD3", "#FF5252", "#33D9B2")
V(net)$cluster <- twomoon$clusters
E(net)$color <- apply(as.data.frame(get.edgelist(net)), 1,
    function(x) ifelse(V(net)$cluster[x[1]] == V(net)$cluster[x[2]],
    colors[V(net)$cluster[x[1]]], "#000000"))
V(net)$color <- c(colors[1], colors[2])[twomoon$clusters]
# plot network
plot(net, layout = twomoon$data, vertex.label = NA, vertex.size = 3)
```

---

### D

**Computes the degree operator from the vector of edge weights.**

<table>
<thead>
<tr>
<th>Description</th>
<th>Computes the degree operator from the vector of edge weights.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Usage</td>
<td>( D(w) )</td>
</tr>
<tr>
<td>Arguments</td>
<td>( w ) vector</td>
</tr>
<tr>
<td>Value</td>
<td>( Dw ) vector</td>
</tr>
</tbody>
</table>

Dstar  Computes the Dstar operator, i.e., the adjoint of the D operator.

Description
Computes the Dstar operator, i.e., the adjoint of the D operator.

Usage
Dstar(w)

Arguments
w vector

Value
Dstar(w) vector

fdr  Computes the false discovery rate between two matrices

Description
Computes the false discovery rate between two matrices

Usage
fdr(Wtrue, West, eps = 1e-04)

Arguments
Wtrue true matrix
West estimated matrix
eps real number such that edges whose values are smaller than eps are not considered in the computation of the fscore

Examples
library(spectralGraphTopology)
X <- L(c(1, 0, 1))
fdr(X, X)
fscore

Computes the fscore between two matrices

Description

Computes the fscore between two matrices

Usage

```r
fscore(Wtrue, West, eps = 1e-04)
```

Arguments

- `Wtrue`: true matrix
- `West`: estimated matrix
- `eps`: real number such that edges whose values are smaller than eps are not considered in the computation of the fscore

Examples

```r
library(spectralGraphTopology)
X <- L(c(1, 0, 1))
fscore(X, X)
```

L

Computes the Laplacian linear operator which maps a vector of weights into a valid Laplacian matrix.

Description

Computes the Laplacian linear operator which maps a vector of weights into a valid Laplacian matrix.

Usage

```r
L(w)
```

Arguments

- `w`: weight vector of the graph

Value

`Lw` the Laplacian matrix
learn_bipartite_graph

Learn a bipartite graph
Learns a bipartite graph on the basis of an observed data matrix

Examples

```r
library(spectralGraphTopology)
Lw <- L(c(1, 0, 1))
Lw
```

Description

Learn a bipartite graph
Learns a bipartite graph on the basis of an observed data matrix

Usage

```r
learn_bipartite_graph(
  S,
  is_data_matrix = FALSE,
  z = 0,
  nu = 10000,
  alpha = 0,
  w0 = "naive",
  m = 7,
  maxiter = 10000,
  abstol = 1e-06,
  reltol = 1e-04,
  record_weights = FALSE,
  verbose = TRUE
)
```

Arguments

- **S**: either a pxp sample covariance/correlation matrix, or a pxn data matrix, where p is the number of nodes and n is the number of features (or data points per node)
- **is_data_matrix**: whether the matrix S should be treated as data matrix or sample covariance matrix
- **z**: the number of zero eigenvalues for the Adjancecy matrix
- **nu**: regularization hyperparameter for the term \(\|A(w) - V \Psi V^\top\|_2\_F\)
- **alpha**: L1 regularization hyperparameter
- **w0**: initial estimate for the weight vector the graph or a string selecting an appropriate method. Available methods are: "qp": finds \(w_0\) that minimizes \(\|\text{ginv}(S) - L(w_0)\|_F\), \(w_0 \geq 0\); "naive": takes \(w_0\) as the negative of the off-diagonal elements of the pseudo inverse, setting to 0 any elements s.t. \(w_0 < 0\)
m

in case is_data_matrix = TRUE, then we build an affinity matrix based on Nie et. al. 2017, where m is the maximum number of possible connections for a given node.

maxiter

the maximum number of iterations

abstol

absolute tolerance on the weight vector w

reltol

relative tolerance on the weight vector w

record_weights

whether to record the edge values at each iteration

verbose

whether to output a progress bar showing the evolution of the iterations

Value

A list containing possibly the following elements:

laplacian

the estimated Laplacian Matrix

adjacency

the estimated Adjacency Matrix

w

the estimated weight vector

psi

optimization variable accounting for the eigenvalues of the Adjacency matrix

V

eigenvectors of the estimated Adjacency matrix

elapsed_time

elapsed time recorded at every iteration

convergence

boolean flag to indicate whether or not the optimization converged

obj_fun

values of the objective function at every iteration in case record_objective = TRUE

negloglike

values of the negative loglikelihood at every iteration in case record_objective = TRUE

w_seq

sequence of weight vectors at every iteration in case record_weights = TRUE

Author(s)

Ze Vinicius and Daniel Palomar

References


Examples

library(spectralGraphTopology)
library(igraph)
library(viridis)
library(corrplot)
set.seed(42)
n1 <- 10
n2 <- 6
n <- n1 + n2
learn_bipartite_k_component_graph

Learns a bipartite k-component graph. Jointly learns the Laplacian and Adjacency matrices of a graph on the basis of an observed data matrix.

Description

Learns a bipartite k-component graph.
Jointly learns the Laplacian and Adjacency matrices of a graph on the basis of an observed data matrix

Usage

```r
learn_bipartite_k_component_graph(
  S,
  is_data_matrix = FALSE,
  z = 0,
  k = 1,
  w0 = "naive",
  m = 7,
  alpha = 0,
  beta = 10000,
  rho = 0.01,
  fix_beta = TRUE,
  beta_max = 1e+06,
  nu = 10000,
  lb = 0,
  ub = 10000,
  maxiter = 10000,
  abstol = 1e-06,
  reltol = 1e-04,
  eigtol = 1e-09,
  record_weights = FALSE,
  record_objective = FALSE,
  verbose = TRUE
)
```

Arguments

- **S**: either a pxp sample covariance/correlation matrix, or a pxn data matrix, where p is the number of nodes and n is the number of features (or data points per node)
- **is_data_matrix**: whether the matrix S should be treated as data matrix or sample covariance matrix
- **z**: the number of zero eigenvalues for the Adjacency matrix
- **k**: the number of components of the graph
- **w0**: initial estimate for the weight vector the graph or a string selecting an appropriate method. Available methods are: "qp": finds w0 that minimizes \( \| \text{ginv}(S) - L(w_0) \|_F \), w0 >= 0; "naive": takes w0 as the negative of the off-diagonal elements of the pseudo inverse, setting to 0 any elements s.t. w0 < 0
- **m**: in case is_data_matrix = TRUE, then we build an affinity matrix based on Nie et. al. 2017, where m is the maximum number of possible connections for a given node
- **alpha**: L1 regularization hyperparameter
- **beta**: regularization hyperparameter for the term \( \| L(w) - U \Lambda U^\top \|_2_F \)
- **rho**: how much to increase (decrease) beta in case fix_beta = FALSE
fix_beta: whether or not to fix the value of beta. In case this parameter is set to false, then beta will increase (decrease) depending whether the number of zero eigenvalues is lesser (greater) than k.

beta_max: maximum allowed value for beta.

nu: regularization hyperparameter for the term \|A(w) - V \Psi V'\|^2_F.

lb: lower bound for the eigenvalues of the Laplacian matrix.

ub: upper bound for the eigenvalues of the Laplacian matrix.

maxiter: the maximum number of iterations.

abstol: absolute tolerance on the weight vector w.

reltol: relative tolerance on the weight vector w.

eigtol: value below which eigenvalues are considered to be zero.

record_weights: whether to record the edge values at each iteration.

record_objective: whether to record the objective function values at each iteration.

verbose: whether to output a progress bar showing the evolution of the iterations.

Value

A list containing possibly the following elements:

laplacian: the estimated Laplacian Matrix.

adjacency: the estimated Adjacency Matrix.

w: the estimated weight vector.

psi: optimization variable accounting for the eigenvalues of the Adjacency matrix.

lambda: optimization variable accounting for the eigenvalues of the Laplacian matrix.

V: eigenvectors of the estimated Adjacency matrix.

U: eigenvectors of the estimated Laplacian matrix.

elapsed_time: elapsed time recorded at every iteration.

beta_seq: sequence of values taken by beta in case fix_beta = FALSE.

convergence: boolean flag to indicate whether or not the optimization converged.

obj_fun: values of the objective function at every iteration in case record_objective = TRUE.

negloglike: values of the negative loglikelihood at every iteration in case record_objective = TRUE.

w_seq: sequence of weight vectors at every iteration in case record_weights = TRUE.

Author(s)

Ze Vinicius and Daniel Palomar.
learn_bipartite_k_component_graph

References


Examples

```r
library(spectralGraphTopology)
library(igraph)
library(viridis)
library(corrplot)
set.seed(42)
w <- c(1, 0, 0, 1, 0, 1) * runif(6)
Laplacian <- block_diag(L(w), L(w))
Atrue <- diag(diag(Laplacian)) - Laplacian
bipartite <- graph_from_adjacency_matrix(Atrue, mode = "undirected", weighted = TRUE)
n <- ncol(Laplacian)
Y <- MASS::mvrnorm(40 * n, rep(0, n), MASS::ginv(Laplacian))
graph <- learn_bipartite_k_component_graph(cov(Y), k = 2, beta = 1e2, nu = 1e2, verbose = FALSE)
graph$adjacency[graph$adjacency < 1e-2] <- 0

# Plot Adjacency matrices: true, noisy, and estimated
corrplot(Atrue / max(Atrue), is.corr = FALSE, method = "square", addgrid.col = NA, tl.pos = "n", cl.cex = 1.25)
corrplot(graph$adjacency / max(graph$adjacency), is.corr = FALSE, method = "square", addgrid.col = NA, tl.pos = "n", cl.cex = 1.25)

# Plot networks
estimated_bipartite <- graph_from_adjacency_matrix(graph$adjacency, mode = "undirected", weighted = TRUE)
V(estimated_bipartite)$type <- rep(c(TRUE, FALSE), 4)
la = layout_as_bipartite(estimated_bipartite)
colors <- viridis(20, begin = 0, end = 1, direction = -1)
c_scale <- colorRamp(colors)
E(estimated_bipartite)$color = apply(c_scale(E(estimated_bipartite)$weight / max(E(estimated_bipartite)$weight)), 1, function(x) rgb(x[1]/255, x[2]/255, x[3]/255))
E(graph$adjacency)[graph$adjacency < 1e-2] <- 0
c_scale <- colorRamp(colors)
E(graph$adjacency)$color = apply(c_scale(E(graph$adjacency)$weight / max(E(graph$adjacency)$weight)), 1, function(x) rgb(x[1]/255, x[2]/255, x[3]/255))

# Plot networks: true and estimated
plot(bipartite, layout = la,
     vertex.color = c("red", "black")[(V(bipartite)$type + 1) - 1],
     vertex.shape = c("square", "circle")[(V(bipartite)$type + 1) - 1],
     vertex.label = NA, vertex.size = 5)
plot(estimated_bipartite, layout = la,
     vertex.color = c("red", "black")[(V(estimated_bipartite)$type + 1) - 1],
     vertex.shape = c("square", "circle")[(V(estimated_bipartite)$type + 1) - 1],
     vertex.label = NA, vertex.size = 5)
```

Learn the Combinatorial Graph Laplacian from data

Learns a graph Laplacian matrix using the Combinatorial Graph Laplacian (CGL) algorithm proposed by Egilmez et. al. (2017)

Description

Learn the Combinatorial Graph Laplacian from data

Learns a graph Laplacian matrix using the Combinatorial Graph Laplacian (CGL) algorithm proposed by Egilmez et. al. (2017)

Usage

```r
learn_combinatorial_graph_laplacian(
  S,
  A_mask = NULL,
  alpha = 0,
  reltol = 1e-05,
  max_cycle = 10000,
  regtype = 1,
  record_objective = FALSE,
  verbose = TRUE
)
```

Arguments

- `S` sample covariance matrix
- `A_mask` binary adjacency matrix of the graph
- `alpha` L1-norm regularization hyperparameter
- `reltol` minimum relative error considered for the stopping criterion
- `max_cycle` maximum number of cycles
- `regtype` type of L1-norm regularization. If `reg_type == 1`, then all elements of the Laplacian matrix will be regularized. If `reg_type == 2`, only the off-diagonal elements will be regularized
- `record_objective` whether or not to record the objective function value at every iteration. Default is FALSE
- `verbose` if TRUE, then a progress bar will be displayed in the console. Default is TRUE

Value

A list containing possibly the following elements

- `laplacian` estimated Laplacian Matrix
learn_graph_sigrep

elapsed_time elapsed time recorded at every iteration
frod_norm relative Frobenius norm between consecutive estimates of the Laplacian matrix
convergence whether or not the algorithm has converged within the tolerance and max number of iterations
obj_fun objective function value at every iteration, in case record_objective = TRUE

References


learn_graph_sigrep Learn graphs from a smooth signal representation approach This function learns a graph from an observed data matrix using the method proposed by Dong (2016).

Description

Learn graphs from a smooth signal representation approach

This function learns a graph from an observed data matrix using the method proposed by Dong (2016).

Usage

learn_graph_sigrep(
  X,
  alpha = 0.001,
  beta = 0.5,
  maxiter = 1000,
  ftol = 1e-04,
  verbose = TRUE
)

Arguments

X a p-by-n data matrix, where p is the number of nodes and n is the number of observations
alpha hyperparameter that controls the importance of the Dirichlet energy penalty
beta hyperparameter that controls the importance of the L2-norm regularization
maxiter maximum number of iterations
ftol relative error on the objective function to be used as the stopping criteria
verbose if TRUE, then a progress bar will be displayed in the console. Default is TRUE
Value

A list containing the following items

- `laplacian` estimated Laplacian Matrix
- `Y` a smoothed approximation of the data matrix `X`
- `convergence` whether or not the algorithm has converged within the tolerance and max number of iterations
- `obj_fun` objective function value at every iteration, in case `record_objective = TRUE`

References


learn_k_component_graph

Learn the Laplacian matrix of a k-component graph Learns a k-component graph on the basis of an observed data matrix. Check out https://mirca.github.io/spectralGraphTopology for code examples.

Description

Learn the Laplacian matrix of a k-component graph

Learns a k-component graph on the basis of an observed data matrix. Check out https://mirca.github.io/spectralGraphTopology for code examples.

Usage

```
learn_k_component_graph(
  S,
  is_data_matrix = FALSE,
  k = 1,
  w0 = "naive",
  lb = 0,
  ub = 10000,
  alpha = 0,
  beta = 10000,
  beta_max = 1e+06,
  fix_beta = TRUE,
  rho = 0.01,
  m = 7,
  eps = 1e-04,
  maxiter = 10000,
  abstol = 1e-06,
  reltol = 1e-04,
)```
learn_k_component_graph

eigtol = 1e-09,
record_objective = FALSE,
record_weights = FALSE,
verbose = TRUE
)

Arguments

S    either a pxp sample covariance/correlation matrix, or a pxn data matrix, where p is the number of nodes and n is the number of features (or data points per node)

is_data_matrix    whether the matrix S should be treated as data matrix or sample covariance matrix

k    the number of components of the graph

w0    initial estimate for the weight vector the graph or a string selecting an appropriate method. Available methods are: "qp": finds w0 that minimizes ||ginv(S) - L(w0)||_F, w0 >= 0; "naive": takes w0 as the negative of the off-diagonal elements of the pseudo inverse, setting to 0 any elements s.t. w0 < 0

lb    lower bound for the eigenvalues of the Laplacian matrix

ub    upper bound for the eigenvalues of the Laplacian matrix

alpha    reweighted 11-norm regularization hyperparameter

beta    regularization hyperparameter for the term ||L(w) - U Lambda U'||^2_F

beta_max    maximum allowed value for beta

fix_beta    whether or not to fix the value of beta. In case this parameter is set to false, then beta will increase (decrease) depending whether the number of zero eigenvalues is lesser (greater) than k

rho    how much to increase (decrease) beta in case fix_beta = FALSE

m    in case is_data_matrix = TRUE, then we build an affinity matrix based on Nie et. al. 2017, where m is the maximum number of possible connections for a given node

eps    small positive constant

maxiter    the maximum number of iterations

abstol    absolute tolerance on the weight vector w

reltol    relative tolerance on the weight vector w

eigtol    value below which eigenvalues are considered to be zero

record_objective    whether to record the objective function values at each iteration

record_weights    whether to record the edge values at each iteration

verbose    whether to output a progress bar showing the evolution of the iterations
Value

A list containing possibly the following elements:

- laplacian: the estimated Laplacian Matrix
- adjacency: the estimated Adjacency Matrix
- w: the estimated weight vector
- lambda: optimization variable accounting for the eigenvalues of the Laplacian matrix
- U: eigenvectors of the estimated Laplacian matrix
- elapsed_time: elapsed time recorded at every iteration
- beta_seq: sequence of values taken by beta in case fix_beta = FALSE
- convergence: boolean flag to indicate whether or not the optimization converged
- obj_fun: values of the objective function at every iteration in case record_objective = TRUE
- negloglike: values of the negative loglikelihood at every iteration in case record_objective = TRUE
- w_seq: sequence of weight vectors at every iteration in case record_weights = TRUE

Author(s)

Ze Vinicius and Daniel Palomar

References


Examples

```r
# design true Laplacian
Laplacian <- rbind(c(1, -1, 0, 0),
                   c(-1, 1, 0, 0),
                   c(0, 0, 1, -1),
                   c(0, 0, -1, 1))
n <- ncol(Laplacian)
# sample data from multivariate Gaussian
Y <- MASS::mvrnorm(n * 500, rep(0, n), MASS::ginv(Laplacian))
# estimate graph on the basis of sampled data
graph <- learn_k_component_graph(cov(Y), k = 2, beta = 10)
graph$laplacian
```
learn_laplacian_gle_admm

Learn the weighted Laplacian matrix of a graph using the ADMM method

Description

Learn the weighted Laplacian matrix of a graph using the ADMM method

Usage

```r
learn_laplacian_gle_admm(
  S,
  A_mask = NULL,
  alpha = 0,
  rho = 1,
  maxiter = 10000,
  reltol = 1e-05,
  record_objective = FALSE,
  verbose = TRUE
)
```

Arguments

- **S**: a pxp sample covariance/correlation matrix
- **A_mask**: the binary adjacency matrix of the graph
- **alpha**: L1 regularization hyperparameter
- **rho**: ADMM convergence rate hyperparameter
- **maxiter**: the maximum number of iterations
- **reltol**: relative tolerance on the Laplacian matrix estimation
- **record_objective**: whether or not to record the objective function. Default is FALSE
- **verbose**: if TRUE, then a progress bar will be displayed in the console. Default is TRUE

Value

A list containing possibly the following elements:

- **Laplacian**: the estimated Laplacian Matrix
- **Adjacency**: the estimated Adjacency Matrix
- **convergence**: boolean flag to indicate whether or not the optimization converged
- **obj_fun**: values of the objective function at every iteration in case record_objective = TRUE
learn_laplacian_gle_mm

Learn the weighted Laplacian matrix of a graph using the MM method

Description

Learn the weighted Laplacian matrix of a graph using the MM method

Usage

```r
learn_laplacian_gle_mm(
  S,
  A_mask = NULL,
  alpha = 0,
  maxiter = 10000,
  reltol = 1e-05,
  record_objective = FALSE,
  verbose = TRUE
)
```

Arguments

- `S`: a pxp sample covariance/correlation matrix
- `A_mask`: the binary adjacency matrix of the graph
- `alpha`: L1 regularization hyperparameter
- `maxiter`: the maximum number of iterations
- `reltol`: relative tolerance on the weight vector w
- `record_objective`: whether or not to record the objective function. Default is FALSE
- `verbose`: if TRUE, then a progress bar will be displayed in the console. Default is TRUE
*learn_smooth_approx_graph*

**Value**

A list containing possibly the following elements:

- `laplacian`: the estimated Laplacian Matrix
- `Adjacency`: the estimated Adjacency Matrix
- `convergence`: boolean flag to indicate whether or not the optimization converged
- `obj_fun`: values of the objective function at every iteration in case `record_objective = TRUE`

**Author(s)**

Ze Vinicius, Jiaxi Ying, and Daniel Palomar

**References**


---

**learn_smooth_approx_graph**

Learns a smooth approximated graph from an observed data matrix. Check out https://mirca.github.io/spectralGraphTopology for code examples.

---

**Description**

Learns a smooth approximated graph from an observed data matrix. Check out https://mirca.github.io/spectralGraphTopology for code examples.

**Usage**

`learn_smooth_approx_graph(Y, m)`

**Arguments**

- `Y`: a p-by-n data matrix, where p is the number of nodes and n is the number of features (or data points per node)
- `m`: the maximum number of possible connections for a given node used to build an affinity matrix

**Value**

A list containing the following elements:

- `laplacian`: the estimated Laplacian Matrix
learn_smooth_graph

**Author(s)**

Ze Vinicius and Daniel Palomar

**References**


---

**learn_smooth_graph**

*Learn a graph from smooth signals This function learns a connected graph given an observed signal matrix using the method proposed by Kalofolias (2016).*

**Description**

Learn a graph from smooth signals

This function learns a connected graph given an observed signal matrix using the method proposed by Kalofolias (2016).

**Usage**

```r
learn_smooth_graph(
  X,
  alpha = 0.01,
  beta = 1e-04,
  step_size = 0.01,
  maxiter = 1000,
  tol = 1e-04
)
```

**Arguments**

- `X`: a p-by-n data matrix, where p is the number of nodes and n is the number of observations
- `alpha`: hyperparameter that controls the importance of the Dirichlet energy penalty
- `beta`: hyperparameter that controls the importance of the L2-norm regularization
- `step_size`: learning rate
- `maxiter`: maximum number of iterations
- `tol`: relative tolerance used as stopping criteria

**References**

Lstar

Computes the Lstar operator.

Description

Computes the Lstar operator.

Usage

Lstar(M)

Arguments

M matrix

Value

w vector

npv

Computes the negative predictive value between two matrices

Description

Computes the negative predictive value between two matrices

Usage

npv(Wtrue, West, eps = 1e-04)

Arguments

Wtrue true matrix
West estimated matrix
eps real number such that edges whose values are smaller than eps are not considered in the computation of the fscore

Examples

library(spectralGraphTopology)
X <- L(c(1, 0, 1))
npv(X, X)
**recall**

*Computes the recall between two matrices*

**Description**

Computes the recall between two matrices

**Usage**

```
recall(Wtrue, West, eps = 1e-04)
```

**Arguments**

- `Wtrue`: true matrix
- `West`: estimated matrix
- `eps`: real number such that edges whose values are smaller than `eps` are not considered in the computation of the f-score

**Examples**

```
library(spectralGraphTopology)
X <- L(c(1, 0, 1))
recall(X, X)
```

---

**relative_error**

*Computes the relative error between the true and estimated matrices*

**Description**

Computes the relative error between the true and estimated matrices

**Usage**

```
relative_error(West, Wtrue)
```

**Arguments**

- `West`: estimated matrix
- `Wtrue`: true matrix

**Examples**

```
library(spectralGraphTopology)
X <- L(c(1, 0, 1))
relative_error(X, X)
```
**specificity**

*Computes the specificity between two matrices*

**Description**

Computes the specificity between two matrices

**Usage**

```r
specificity(Wtrue, West, eps = 1e-04)
```

**Arguments**

- `Wtrue` : true matrix
- `West` : estimated matrix
- `eps` : real number such that edges whose values are smaller than eps are not considered in the computation of the fscore

**Examples**

```r
library(spectralGraphTopology)
X <- L(c(1, 0, 1))
specificity(X, X)
```
Index

A, 3, 3
accuracy, 4
Astar, 4
block_diag, 5
cluster_k_component_graph, 3, 5
D, 7
Dstar, 8
fdr, 8
fscore, 9
L, 3, 9
learn_bipartite_graph, 3, 10
learn_bipartite_k_component_graph, 3, 12
learn_combinatorial_graph_laplacian, 16
learn_graph_sigrep, 17
learn_k_component_graph, 3, 18
learn_laplacian_gle_admm, 3, 21
learn_laplacian_gle_mm, 3, 22
learn_smooth_approx_graph, 23
learn_smooth_graph, 24
Lstar, 25

npv, 25

recall, 26
relative_error, 26

specificity, 27
spectralGraphTopology-package, 2