Package ‘finbipartite’

February 22, 2023

Title Learning Bipartite Graphs: Heavy Tails and Multiple Components

Version 0.1.0

Date 2023-02-02

Description Learning bipartite and k-component bipartite graphs from financial datasets. This package contains implementations of the algorithms described in the paper: Cardoso JVM, Ying J, and Palomar DP (2022). <https://openreview.net/pdf?id=WNSyF9qZaMd> ``Learning bipartite graphs: heavy tails and multiple components, Advances in Neural Informations Processing Systems'' (NeurIPS).

URL https://github.com/convexfi/bipartite/

BugReports https://github.com/convexfi/bipartite/issues

License GPL-3

Encoding UTF-8

Depends spectralGraphTopology, quadprog

Imports MASS, stats, progress, mvtnorm, CVXR

Suggests testthat, igraph,

RoxygenNote 7.1.1

NeedsCompilation no

Author Ze Vinicius [cre, aut]

Maintainer Ze Vinicius <jvmirca@gmail.com>

Repository CRAN

Date/Publication 2023-02-22 14:40:06 UTC

R topics documented:

learn_bipartite_graph_nie
Laplacian matrix of a k-component bipartite graph via Nie's method Computes the Laplacian matrix of a bipartite graph on the basis of an observed similarity matrix.

Description

Laplacian matrix of a k-component bipartite graph via Nie's method

Computes the Laplacian matrix of a bipartite graph on the basis of an observed similarity matrix.

Usage

```
learn_bipartite_graph_nie(
  S,
  r,
  q,
  k,
  learning_rate = 1e-04,
  eta = 1,
  maxiter = 1000,
  reltol = 1e-06,
  verbose = TRUE,
  record_objective = FALSE
)
```

Arguments

`S`	a p x p similarity matrix, where p is the number of nodes in the graph.
`r`	number of nodes in the objects set.
`q`	number of nodes in the classes set.
`k`	number of components of the graph.
`learning_rate`	gradient descent parameter.
`eta`	rank constraint hyperparameter.
`maxiter`	maximum number of iterations.
`reltol`	relative tolerance as a convergence criteria.
`verbose`	whether or not to show a progress bar during the iterations.
`record_objective`	whether or not to record the objective function value during iterations.

Value

A list containing possibly the following elements:

laplacian	estimated Laplacian matrix
adjacency	estimated adjacency matrix
B	estimated graph weights matrix
maxiter	number of iterations taken to reach convergence
convergence	boolean flag to indicate whether or not the optimization converged
obj_fun	objective function value per iteration

References

Feiping Nie, Xiaoqian Wang, Cheng Deng, Heng Huang. "Learning A Structured Optimal Bipartite Graph for Co-Clustering". Advances in Neural Information Processing Systems (NIPS 2017)

Examples

```
library(finbipartite)
library(igraph)
set.seed(42)
r <- 50
q <- 5
p <- r + q

bipartite <- sample_bipartite(r, q, type="Gnp", p = 1, directed=FALSE)
# randomly assign edge weights to connected nodes
E(bipartite)$weight <- 1
Lw <- as.matrix(laplacian_matrix(bipartite))
B <- -Lw[1:r, (r+1):p]
B[,] <- runif(length(B))
B <- B / rowSums(B)
# utils functions
from_B_to_laplacian <- function(B) {
  A <- from_B_to_adjacency(B)
  return(diag(rowSums(A)) - A)
}

from_B_to_adjacency <- function(B) {
  r <- nrow(B)
  q <- ncol(B)
  zeros_rxr <- matrix(0, r, r)
  zeros_qxq <- matrix(0, q, q)
  return(rbind(cbind(zeros_rxr, B), cbind(t(B), zeros_qxq)))
}
Ltrue <- from_B_to_laplacian(B)
X <- MASS::mvrnorm(100*p, rep(0, p), MASS::ginv(Ltrue))
S <- cov(X)
bipartite_graph <- learn_bipartite_graph_nie(S = S,
                                             r = r,
                                             q = q,
```

```
k = 1,
learning_rate = 5e-1,
eta = 0,
verbose=FALSE)
```

learn_connected_bipartite_graph_pgd

Laplacian matrix of a connected bipartite graph with Gaussian data Computes the Laplacian matrix of a bipartite graph on the basis of an observed data matrix.

Description

Laplacian matrix of a connected bipartite graph with Gaussian data

Computes the Laplacian matrix of a bipartite graph on the basis of an observed data matrix.

Usage

```
learn_connected_bipartite_graph_pgd(
  S,
  r,
  q,
  init = "naive",
  learning_rate = 1e-04,
  maxiter = 1000,
  reltol = 1e-05,
  verbose = TRUE,
  record_objective = FALSE,
  backtrack = TRUE
)
```

Arguments

S	a p x p covariance matrix, where p is the number of nodes in the graph.
r	number of nodes in the objects set.
q	number of nodes in the classes set.
init	string denoting how to compute the initial graph.
learning_rate	gradient descent parameter.
maxiter	maximum number of iterations.
reltol	relative tolerance as a convergence criteria.
verbose	whether or not to show a progress bar during the iterations.
record_objective	whether or not to record the objective function value during iterations.
backtrack	whether or not to optimize the learning rate via backtracking.

Value

A list containing possibly the following elements:

`laplacian`	estimated Laplacian matrix
`adjacency`	estimated adjacency matrix
`B`	estimated graph weights matrix
`maxiter`	number of iterations taken to reach convergence
`convergence`	boolean flag to indicate whether or not the optimization converged
`lr_seq`	learning rate value per iteration
`obj_seq`	objective function value per iteration
`elapsed_time`	time taken per iteration until convergence is reached

Examples

```
library(finbipartite)
library(igraph)
set.seed(42)
r <- 50
q <- 5
p <- r + q

bipartite <- sample_bipartite(r, q, type="Gnp", p = 1, directed=FALSE)
# randomly assign edge weights to connected nodes
E(bipartite)$weight <- 1
Lw <- as.matrix(laplacian_matrix(bipartite))
B <- -Lw[1:r, (r+1):p]
B[,] <- runif(length(B))
B <- B / rowSums(B)
# utils functions
from_B_to_laplacian <- function(B) {
  A <- from_B_to_adjacency(B)
  return(diag(rowSums(A)) - A)
}

from_B_to_adjacency <- function(B) {
  r <- nrow(B)
  q <- ncol(B)
  zeros_rxr <- matrix(0, r, r)
  zeros_qxq <- matrix(0, q, q)
  return(rbind(cbind(zeros_rxr, B), cbind(t(B), zeros_qxq)))
}
Ltrue <- from_B_to_laplacian(B)
X <- MASS::mvrnorm(100*p, rep(0, p), MASS::ginv(Ltrue))
S <- cov(X)
bipartite_graph <- learn_connected_bipartite_graph_pgd(S = S,
                                                       r = r,
                                                       q = q,
                                                       verbose=FALSE)
```

learn_heavy_tail_bipartite_graph_pgd
Laplacian matrix of a connected bipartite graph with heavy-tailed data Computes the Laplacian matrix of a bipartite graph on the basis of an observed data matrix whose distribution is assumed to be Student-t.

Description

Laplacian matrix of a connected bipartite graph with heavy-tailed data

Computes the Laplacian matrix of a bipartite graph on the basis of an observed data matrix whose distribution is assumed to be Student-t.

Usage

```
learn_heavy_tail_bipartite_graph_pgd(
  X,
  r,
  q,
  nu = 2.001,
  learning_rate = 1e-04,
  maxiter = 1000,
  reltol = 1e-05,
  init = "default",
  verbose = TRUE,
  record_objective = FALSE,
  backtrack = TRUE
)
```

Arguments

X	a n x p data matrix, where p is the number of nodes in the graph and n is the number of observations.
r	number of nodes in the objects set.
q	number of nodes in the classes set.
nu	degrees of freedom of the Student-t distribution.
learning_rate	gradient descent parameter.
maxiter	maximum number of iterations.
reltol	relative tolerance as a convergence criteria.
init	string denoting how to compute the initial graph or a r x q matrix with initial graph weights.
verbose	whether or not to show a progress bar during the iterations.
record_objective	whether or not to record the objective function value during iterations.
backtrack	whether or not to optimize the learning rate via backtracking.

Value

A list containing possibly the following elements:

laplacian	estimated Laplacian matrix
adjacency	estimated adjacency matrix
B	estimated graph weights matrix
maxiter	number of iterations taken to reach convergence
convergence	boolean flag to indicate whether or not the optimization converged
lr_seq	learning rate value per iteration
obj_seq	objective function value per iteration
elapsed_time	time taken per iteration until convergence is reached

Examples

```
library(finbipartite)
library(igraph)
set.seed(42)
r <- 50
q <- 5
p <- r + q

bipartite <- sample_bipartite(r, q, type="Gnp", p = 1, directed=FALSE)
# randomly assign edge weights to connected nodes
E(bipartite)$weight <- 1
Lw <- as.matrix(laplacian_matrix(bipartite))
B <- -Lw[1:r, (r+1):p]
B[,] <- runif(length(B))
B <- B / rowSums(B)
# utils functions
from_B_to_laplacian <- function(B) {
  A <- from_B_to_adjacency(B)
  return(diag(rowSums(A)) - A)
}

from_B_to_adjacency <- function(B) {
  r <- nrow(B)
  q <- ncol(B)
  zeros_rxr <- matrix(0, r, r)
  zeros_qxq <- matrix(0, q, q)
  return(rbind(cbind(zeros_rxr, B), cbind(t(B), zeros_qxq)))
}
Ltrue <- from_B_to_laplacian(B)
X <- MASS::mvrnorm(100*p, rep(0, p), MASS::ginv(Ltrue))
bipartite_graph <- learn_heavy_tail_bipartite_graph_pgd(X = X,
                                                        r = r,
                                                        q = q,
                                                        nu = 1e2,
                                                        verbose=FALSE)
```

learn_heavy_tail_kcomp_bipartite_graph
Laplacian matrix of a k-component bipartite graph with heavy-tailed data Computes the Laplacian matrix of a k-component bipartite graph on the basis of an observed data matrix whose distribution is assumed to be Student-t.

Description

Laplacian matrix of a k-component bipartite graph with heavy-tailed data

Computes the Laplacian matrix of a k-component bipartite graph on the basis of an observed data matrix whose distribution is assumed to be Student-t.

Usage

```
learn_heavy_tail_kcomp_bipartite_graph(
  X,
  r,
  q,
  k,
  nu = 2.001,
  rho = 1,
  learning_rate = 1e-04,
  maxiter = 1000,
  reltol = 1e-05,
  init = "default",
  verbose = TRUE,
  record_objective = FALSE
)
```

Arguments

X	a n x p data matrix, where p is the number of nodes in the graph and n is the number of observations.
r	number of nodes in the objects set.
q	number of nodes in the classes set.
k	number of components of the graph.
nu	degrees of freedom of the Student-t distribution.
rho	ADMM hyperparameter.
learning_rate	gradient descent parameter.
maxiter	maximum number of iterations.
reltol	relative tolerance as a convergence criteria.
init	string denoting how to compute the initial graph or a r x q matrix with initial graph weights.

verbose whether or not to show a progress bar during the iterations.

record_objective
  whether or not to record the objective function value during iterations.

Value

A list containing possibly the following elements:

laplacian	estimated Laplacian matrix
adjacency	estimated adjacency matrix
B	estimated graph weights matrix
maxiter	number of iterations taken to reach convergence
convergence	boolean flag to indicate whether or not the optimization converged
dual_residual	dual residual value per iteration
primal_residual	primal residual value per iteration
aug_lag	augmented Lagrangian value per iteration
rho_seq	constraint relaxation hyperparameter value per iteration
elapsed_time	time taken per iteration until convergence is reached

Examples

```
library(finbipartite)
library(igraph)
set.seed(42)
r <- 50
q <- 5
p <- r + q

bipartite <- sample_bipartite(r, q, type="Gnp", p = 1, directed=FALSE)
# randomly assign edge weights to connected nodes
E(bipartite)$weight <- 1
Lw <- as.matrix(laplacian_matrix(bipartite))
B <- -Lw[1:r, (r+1):p]
B[,] <- runif(length(B))
B <- B / rowSums(B)
# utils functions
from_B_to_laplacian <- function(B) {
  A <- from_B_to_adjacency(B)
  return(diag(rowSums(A)) - A)
}

from_B_to_adjacency <- function(B) {
  r <- nrow(B)
  q <- ncol(B)
  zeros_rxr <- matrix(0, r, r)
  zeros_qxq <- matrix(0, q, q)
  return(rbind(cbind(zeros_rxr, B), cbind(t(B), zeros_qxq)))
}
```

```
Ltrue <- from_B_to_laplacian(B)
X <- MASS::mvrnorm(100*p, rep(0, p), MASS::ginv(Ltrue))
bipartite_graph <- learn_heavy_tail_kcomp_bipartite_graph(X = X,
                                                          r = r,
                                                          q = q,
                                                          k = 1,
                                                          nu = 1e2,
                                                          verbose=FALSE)
```

Index