Package 'CASCORE'

May 7, 2026

Type Package

Title Covariate Assisted Spectral Clustering on Ratios of Eigenvectors

Version 0.1.2

Author Yaofang Hu [aut, cre],
Wanjie Wang [aut]

Maintainer Yaofang Hu <yaofangh@smu.edu>

Description Functions for implementing the novel algorithm CASCORE, which is designed to detect latent community structure in graphs with node covariates. This algorithm can handle models such as the covariate-assisted degree corrected stochastic block model (CADCSBM). CASCORE specifically addresses the disagreement between the community structure inferred from the adjacency information and the community structure inferred from the covariate information. For more detailed information, please refer to the reference paper: Yaofang Hu and Wanjie Wang (2022) <doi:10.48550/arXiv.2306.15616>.
In addition to CASCORE, this package includes several classical community detection algorithms that are compared to CASCORE in our paper. These algorithms are: Spectral Clustering On Ratios-of Eigenvectors (SCORE), normalized PCA, ordinary PCA, network-based clustering, covariates-based clustering and covariate-assisted spectral clustering (CASC). By providing these additional algorithms, the package enables users to compare their performance with CASCORE in community detection tasks.

Imports stats, pracma

License GPL-2

Encoding UTF-8

URL https://arxiv.org/abs/2306.15616

RoxygenNote 7.2.3

Suggests testthat, igraph

NeedsCompilation no

Repository CRAN

Date/Publication 2023-07-02 05:00:02 UTC

Contents

`ADMM` *Penalized Optimization Framework for Community Detection in Networks with Covariates.*

Description

Semidefinite programming for optimizing the inner product between combined network and the solution matrix.

Usage

```
ADMM(
  Adj,
  Covariate,
  lambda,
  K,
  alpha,
  rho,
  TT,
  tol,
  quiet = NULL,
  report_interval = NULL,
  r = NULL
)
```

Arguments

`Adj`	A 0/1 adjacency matrix.
`Covariate`	A covariate matrix. The rows correspond to nodes and the columns correspond to covariates.
`lambda`	A tuning parameter to weigh the covariate matrix.
`K`	A positive integer, indicating the number of underlying communities in graph `Adj`.
`alpha`	A number. The elementwise upper bound in the SDP.

rho	The learning rate of ADMM.
TT	The maximum of iteration.
tol	The tolerance for stopping criterion.
quiet	An optional inoput. Whether to print result at each step.
report_interval	An optional inoput. The frequency to print intermediate result.
r	An optional inoput. The expected rank of the solution, leave NULL if no constraint is required.

Details

ADMM is proposed in *Covariate Regularized Community Detection in Sparse Graphs* of Yan & Sarkar (2021). *ADMM* relies on semidefinite programming (SDP) relaxations for detecting the community structure in sparse networks with covariates.

Value

estall	A lavel vector.

References

Yan, B., & Sarkar, P. (2021). *Covariate Regularized Community Detection in Sparse Graphs*. *Journal of the American Statistical Association, 116(534), 734-745*.
doi:10.1080/01621459.2019.1706541

Examples

```
# Simulate the Network
n = 10; K = 2;
theta = 0.4 + (0.45-0.05)*(seq(1:n)/n)^2; Theta = diag(theta);
P  = matrix(c(0.8, 0.2, 0.2, 0.8), byrow = TRUE, nrow = K)
set.seed(2022)
l = sample(1:K, n, replace=TRUE); # node labels
Pi = matrix(0, n, K) # label matrix
for (k in 1:K){
  Pi[l == k, k] = 1
}
Omega = Theta %*% Pi %*% P %*% t(Pi) %*% Theta;
Adj = matrix(runif(n*n, 0, 1), nrow = n);
Adj = Omega - Adj;
Adj = 1*(Adj >= 0)
diag(Adj) = 0
Adj[lower.tri(Adj)] = t(Adj)[lower.tri(Adj)]
caseno = 4; Nrange = 10; Nmin = 10; prob1 = 0.9; p = n*4;
Q = matrix(runif(p*K, 0, 1), nrow = p, ncol = K)
Q = sweep(Q,2,colSums(Q),`/`)
W = matrix(0, nrow = n, ncol = K);
for(jj in 1:n) {
  if(runif(1) <= prob1) {W[jj, 1:K] = Pi[jj, ];}
```

```
  else W[jj, sample(K, 1)] = 1;
}
W = t(W)
D0 = Q %*% W
X = matrix(0, n, p)
N = switch(caseno, rep(100, n), rep(100, n), round(runif(n)*Nrange+ Nmin),
  round(runif(n)* Nrange+Nmin))
for (i in 1: ncol(D0)){
  X[i, ] = rmultinom(1, N[i], D0[, i])
}
ADMM(Adj, X, lambda = 0.2, K = K, alpha = 0.5, rho = 2, TT = 100, tol = 5)
```

CASC *Covariate Assisted Spectral Clustering.*

Description

CASC clusters graph nodes by applying spectral clustering with the assistance from node covariates.

Usage

```
CASC(Adj, Covariate, K, alphan = 5, itermax = 100, startn = 10)
```

Arguments

Adj	A 0/1 adjacency matrix.
Covariate	A covariate matrix. The rows correspond to nodes and the columns correspond to covariates.
K	A positive integer, indicating the number of underlying communities in graph Adj.
alphan	A tuning parameter to balance between the contributions of the graph and the covariates.
itermax	k-means parameter, indicating the maximum number of iterations allowed. The default value is 100.
startn	k-means parameter. If centers is a number, how many random sets should be chosen? The default value is 10.

Details

CASC is a community detection algorithm for networks with node covariates, proposed in *Covariate-assisted spectral clustering* of Binkiewicz, et al. (2017). *CASC* applies k-means on the first K leading eigenvectors of the balanced matrix between the Laplacian matrix and the covariate matrix.

Value

estall	A lavel vector.

References

Binkiewicz, N., Vogelstein, J. T., & Rohe, K. (2017). *Covariate-assisted spectral clustering*. *Biometrika, 104(2), 361-377.*
doi:10.1093/biomet/asx008

Examples

```
# Simulate the Network
n = 10; K = 2;
theta = 0.4 + (0.45-0.05)*(seq(1:n)/n)^2; Theta = diag(theta);
P  = matrix(c(0.8, 0.2, 0.2, 0.8), byrow = TRUE, nrow = K)
set.seed(2022)
l = sample(1:K, n, replace=TRUE); # node labels
Pi = matrix(0, n, K) # label matrix
for (k in 1:K){
  Pi[l == k, k] = 1
}
Omega = Theta %*% Pi %*% P %*% t(Pi) %*% Theta;
Adj = matrix(runif(n*n, 0, 1), nrow = n);
Adj = Omega - Adj;
Adj = 1*(Adj >= 0)
diag(Adj) = 0
Adj[lower.tri(Adj)] = t(Adj)[lower.tri(Adj)]
caseno = 4; Nrange = 10; Nmin = 10; prob1 = 0.9; p = n*4;
Q = matrix(runif(p*K, 0, 1), nrow = p, ncol = K)
Q = sweep(Q,2,colSums(Q),`/`)
W = matrix(0, nrow = n, ncol = K);
for(jj in 1:n) {
  if(runif(1) <= prob1) {W[jj, 1:K] = Pi[jj, ];}
  else W[jj, sample(K, 1)] = 1;
}
W = t(W)
D0 = Q %*% W
X = matrix(0, n, p)
N = switch(caseno, rep(100, n), rep(100, n), round(runif(n)*Nrange+ Nmin),
  round(runif(n)* Nrange+Nmin))
for (i in 1: ncol(D0)){
  X[i, ] = rmultinom(1, N[i], D0[, i])
}
CASC(Adj, X, 2)
```

CASCORE *Covariate Assisted Spectral Clustering on Ratios of Eigenvectors.*

Description

Using ratios-of-eigenvectors to detect underlying communities in networks with node covariates.

Usage

```
CASCORE(
  Adj,
  Covariate,
  K,
  alpha = NULL,
  alphan = 5,
  itermax = 100,
  startn = 10
)
```

Arguments

`Adj`	A 0/1 adjacency matrix.
`Covariate`	A covariate matrix. The rows correspond to nodes and the columns correspond to covariates.
`K`	A positive integer, indicating the number of underlying communities in graph `Adj`.
`alpha`	A numeric vector, each element of which is a tuning parameter to weigh the covariate matrix.
`alphan`	The number of candidates α. The default number is 5.
`itermax`	`k-means` parameter, indicating the maximum number of iterations allowed. The default value is 100.
`startn`	`k-means` parameter. If centers is a number, how many random sets should be chosen? The default value is 10.

Details

CASCORE is fully established in *Network-Adjusted Covariates for Community Detection* of Hu & Wang (2023). *CASCORE* detects the latent community structure under the covariate assisted degree corrected stochastic block model (CADCSBM), and it allows the disagreement between the community structures indicated in the graph and the covariates, respectively. `K-means` is applied on the entry-wise ratios between first leading eigenvector and each of the other K leading eigenvectors of the combined matrix of the adjacency matrix and the covariate matrix, to reveal the underlying memberships.

Value

`estall`	A lavel vector

.

References

Hu, Y., & Wang, W. (2023) *Network-AdjustedCovariatesforCommunity Detection*, https://arxiv.org/abs/2306.15616

Examples

```
# Simulate the Network
n = 10; K = 2;
theta = 0.4 + (0.45-0.05)*(seq(1:n)/n)^2; Theta = diag(theta);
P  = matrix(c(0.8, 0.2, 0.2, 0.8), byrow = TRUE, nrow = K)
set.seed(2022)
l = sample(1:K, n, replace=TRUE); # node labels
Pi = matrix(0, n, K) # label matrix
for (k in 1:K){
  Pi[l == k, k] = 1
}
Omega = Theta %*% Pi %*% P %*% t(Pi) %*% Theta;
Adj = matrix(runif(n*n, 0, 1), nrow = n);
Adj = Omega - Adj;
Adj = 1*(Adj >= 0)
diag(Adj) = 0
Adj[lower.tri(Adj)] = t(Adj)[lower.tri(Adj)]
caseno = 4; Nrange = 10; Nmin = 10; prob1 = 0.9; p = n*4;
Q = matrix(runif(p*K, 0, 1), nrow = p, ncol = K)
Q = sweep(Q,2,colSums(Q),`/`)
W = matrix(0, nrow = n, ncol = K);
for(jj in 1:n) {
  if(runif(1) <= prob1) {W[jj, 1:K] = Pi[jj, ];}
  else W[jj, sample(K, 1)] = 1;
}
W = t(W)
D0 = Q %*% W
X = matrix(0, n, p)
N = switch(caseno, rep(100, n), rep(100, n), round(runif(n)*Nrange+ Nmin),
  round(runif(n)* Nrange+Nmin))
for (i in 1: ncol(D0)){
  X[i, ] = rmultinom(1, N[i], D0[, i])
}
CASCORE(Adj, X, 2)
```

Cov_based *Covariates-based Clustering.*

Description

Covariates-based Clustering is a spectral clustering method that focuses solely on the covariates structure of a network. It employs k-means on the first K leading eigenvectors of the weighted cogariates matrix of a graph, with each eigenvector normalized to have unit magnitude.

Usage

```
Cov_based(Adj, K, tau = NULL, itermax = NULL, startn = NULL)
```

Arguments

Adj	A 0/1 adjacency matrix.
K	A positive integer, indicating the number of underlying communities in graph Adj.
tau	An optional tuning parameter, the default value is the mean of adajacency matrix.
itermax	k-means parameter, indicating the maximum number of iterations allowed. The default value is 100.
startn	k-means parameter. If centers is a number, how many random sets should be chosen? The default value is 10.

Value

A label vector.

Examples

```
# Simulate the Network
n = 10; K = 2;
theta = 0.4 + (0.45-0.05)*(seq(1:n)/n)^2; Theta = diag(theta);
P  = matrix(c(0.8, 0.2, 0.2, 0.8), byrow = TRUE, nrow = K)
set.seed(2022)
l = sample(1:K, n, replace=TRUE); # node labels
Pi = matrix(0, n, K) # label matrix
for (k in 1:K){
  Pi[l == k, k] = 1
}
Omega = Theta %*% Pi %*% P %*% t(Pi) %*% Theta;
Adj = matrix(runif(n*n, 0, 1), nrow = n);
Adj = Omega - Adj;
Adj = 1*(Adj >= 0)
diag(Adj) = 0
Adj[lower.tri(Adj)] = t(Adj)[lower.tri(Adj)]
Cov_based(Adj, 2)
```

Net_based *Network-based Clustering.*

Description

Network-based Clustering is a spectral clustering method that focuses solely on the topological structure of a network. It employs k-means on the first K leading eigenvectors of the weighted adjacency matrix of a graph, with each eigenvector normalized to have unit magnitude.

Usage

```
Net_based(Adj, K, tau = NULL, itermax = NULL, startn = NULL)
```

Arguments

Adj	A 0/1 adjacency matrix.
K	A positive integer, indicating the number of underlying communities in graph Adj.
tau	An optional tuning parameter, the default value is the mean of adajacency matrix.
itermax	k-means parameter, indicating the maximum number of iterations allowed. The default value is 100.
startn	k-means parameter. If centers is a number, how many random sets should be chosen? The default value is 10.

Value

A label vector.

Examples

```
# Simulate the Network
n = 10; K = 2;
theta = 0.4 + (0.45-0.05)*(seq(1:n)/n)^2; Theta = diag(theta);
P  = matrix(c(0.8, 0.2, 0.2, 0.8), byrow = TRUE, nrow = K)
set.seed(2022)
l = sample(1:K, n, replace=TRUE); # node labels
Pi = matrix(0, n, K) # label matrix
for (k in 1:K){
  Pi[l == k, k] = 1
}
Omega = Theta %*% Pi %*% P %*% t(Pi) %*% Theta;
Adj = matrix(runif(n*n, 0, 1), nrow = n);
Adj = Omega - Adj;
Adj = 1*(Adj >= 0)
diag(Adj) = 0
Adj[lower.tri(Adj)] = t(Adj)[lower.tri(Adj)]
Net_based(Adj, 2)
```

---

nPCA *Normalized Principle Component Analysis.*

Description

Normalized Principle Component Analysis (nPCA), also known as spectral clustering on the graph Laplacian, is a classical spectral clustering method that applies k-means on the first K leading (unit-norm) eigenvectors of the degree-corrected normalized graph laplacian.

Usage

```
nPCA(Adj, K, tau = NULL, itermax = 100, startn = 10)
```

Arguments

Adj	A 0/1 adjacency matrix.
K	A positive integer, indicating the number of underlying communities in graph Adj.
tau	An optional regularization parameter for suitable degree normalization. The default value is the average degree of graph Adj.
itermax	k-means parameter, indicating the maximum number of iterations allowed. The default value is 100.
startn	k-means parameter. If centers is a number, how many random sets should be chosen? The default value is 10.

Value

estall	A lavel vector.

References

Chung, F. R., & Graham, F. C. (1997). *Spectral graph theory (Vol. 92). American Mathematical Soc.*.

Examples

```
# Simulate the Network
n = 10; K = 2;
theta = 0.4 + (0.45-0.05)*(seq(1:n)/n)^2; Theta = diag(theta);
P  = matrix(c(0.8, 0.2, 0.2, 0.8), byrow = TRUE, nrow = K)
set.seed(2022)
l = sample(1:K, n, replace=TRUE); # node labels
Pi = matrix(0, n, K) # label matrix
for (k in 1:K){
  Pi[l == k, k] = 1
}
Omega = Theta %*% Pi %*% P %*% t(Pi) %*% Theta;
Adj = matrix(runif(n*n, 0, 1), nrow = n);
Adj = Omega - Adj;
Adj = 1*(Adj >= 0)
diag(Adj) = 0
Adj[lower.tri(Adj)] = t(Adj)[lower.tri(Adj)]
nPCA(Adj, 2)
```

---

oPCA *Ordinary Principle Component Analysis.*

Description

Ordinary Principle Component Analysis (oPCA), also known as spectral clustering on the adjacency matrix is a classical spectral clustering method that applies k-means on the first K leading (unit-norm) eigenvectors of the adjacency matrix of a graph.

Usage

```
oPCA(Adj, K, itermax = 100, startn = 10)
```

Arguments

Adj	A 0/1 adjacency matrix.
K	A positive integer, indicating the number of underlying communities in graph Adj.
itermax	k-means parameter, indicating the maximum number of iterations allowed. The default value is 100.
startn	k-means parameter. If centers is a number, how many random sets should be chosen? The default value is 10.

Value

estall	A lavel vector.

References

Chung, F. R., & Graham, F. C. (1997). *Spectral graph theory (Vol. 92). American Mathematical Soc.*.

Examples

```
# Simulate the Network
n = 10; K = 2;
theta = 0.4 + (0.45-0.05)*(seq(1:n)/n)^2; Theta = diag(theta);
P  = matrix(c(0.8, 0.2, 0.2, 0.8), byrow = TRUE, nrow = K)
set.seed(2022)
l = sample(1:K, n, replace=TRUE); # node labels
Pi = matrix(0, n, K) # label matrix
for (k in 1:K){
  Pi[l == k, k] = 1
}
Omega = Theta %*% Pi %*% P %*% t(Pi) %*% Theta;
Adj = matrix(runif(n*n, 0, 1), nrow = n);
Adj = Omega - Adj;
Adj = 1*(Adj >= 0)
diag(Adj) = 0
Adj[lower.tri(Adj)] = t(Adj)[lower.tri(Adj)]
oPCA(Adj, 2)
```

SCORE *Spectral Clustering On Ratios-of-Eigenvectors.*

Description

Using ratios-of-eigenvectors to detect underlying communities.

Usage

```
SCORE(G, K, itermax = NULL, startn = NULL)
```

Arguments

G	A 0/1 adjacency matrix of a connected graph.
K	A positive integer, indicating the number of underlying communities in graph G.
itermax	k-means parameter, indicating the maximum number of iterations allowed. The default value is 100.
startn	k-means parameter. If centers is a number, how many random sets should be chosen? The default value is 10.

Details

SCORE is fully established in *Fast community detection by SCORE* of Jin (2015). *SCORE* uses the entry-wise ratios between the first leading eigenvector and each of the other $K-1$ leading eigenvectors for clustering. It is noteworthy that SCORE only works on connected graphs, in other words, it does not allow for isolated vertices.

Value

estall	A lavel vector.

References

Jin, J. (2015). *Fast community detection by score*. *The Annals of Statistics 43 (1), 57–89.* doi:10.1214/14AOS1265

Examples

```
# Simulate the Network
n = 10; K = 2;
theta = 0.4 + (0.45-0.05)*(seq(1:n)/n)^2; Theta = diag(theta);
P  = matrix(c(0.8, 0.2, 0.2, 0.8), byrow = TRUE, nrow = K)
set.seed(2022)
l = sample(1:K, n, replace=TRUE); # node labels
Pi = matrix(0, n, K) # label matrix
for (k in 1:K){
  Pi[l == k, k] = 1
```

```
}
Omega = Theta %*% Pi %*% P %*% t(Pi) %*% Theta;
Adj = matrix(runif(n*n, 0, 1), nrow = n);
Adj = Omega - Adj;
Adj = 1*(Adj >= 0)
diag(Adj) = 0
Adj[lower.tri(Adj)] = t(Adj)[lower.tri(Adj)]
library(igraph)
is.igraph(Adj) # [1] FALSE
ix = components(graph.adjacency(Adj))
componentLabel = ix$membership
giantLabel = which(componentLabel == which.max(ix$csize))
Giant = Adj[giantLabel, giantLabel]
SCORE(Giant, 2)
```

Index