Package ‘NAC’

December 4, 2023

Type Package

Title Network-Adjusted Covariates for Community Detection

Version 0.1.0

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

Maintainer Yaofang Hu <yaofangh@smu.edu>

Description Incorporating node-level covariates for community detection has gained increasing attention these years. This package provides the function for implementing the novel community detection algorithm known as Network-Adjusted Covariates for Community Detection (NAC), which is designed to detect latent community structure in graphs with node-level information, i.e., covariates. This algorithm can handle models such as the degree-corrected stochastic block model (DCSBM) with covariates. NAC specifically addresses the discrepancy between the community structure inferred from the adjacency information and the community structure inferred from the covariates information. For more detailed information, please refer to the reference paper: Yaofang Hu and Wanjie Wang (2023) <arXiv:2306.15616>. In addition to NAC, this package includes several other existing community detection algorithms that are compared to NAC in the reference paper. These algorithms are Spectral Clustering On Ratios-of Eigenvectors (SCORE), network-based regularized spectral clustering (Net-based), covariate-based spectral clustering (Cov-based), covariate-assisted spectral clustering (CAclustering) and semidefinite programming (SDP).

Imports stats, pracma

License GPL-2

Encoding UTF-8

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

RoxygenNote 7.2.3

Suggests testthat, igraph

Depends R (>= 4.2.2.0)

NeedsCompilation no

Repository CRAN

Date/Publication 2023-12-04 16:40:15 UTC

R topics documented:

`CAclustering` *Covariate Assisted Spectral Clustering.*

Description

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

Usage

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

Arguments

`Adj`	An $n \times n$ symmetric adjacency matrix with diagonals being 0 and positive entries being 1.
`Covariate`	An $n \times p$ covariate matrix. The rows correspond to nodes and the columns correspond to covariates.
`K`	A positive integer which is no larger than n. This is the predefined number of communities.
`alphan`	The number of candidate α's to try within the range $(\alpha_{min}, \alpha_{max})$ given in Binkiewicz et al. (2017). An optimal α is expected to achieve a balance between L_τ and X.
`itermax`	`k-means` parameter, indicating the maximum number of iterations allowed. The default value is 100.
`startn`	`k-means` parameter. The number of times the algorithm should be run with different initial centroids. The default value is 10.

Details

`CAclustering` is an algorithm designed for community detection in networks with node covariates, as introduced in the paper *Covariate-assisted spectral clustering* of Binkiewicz et al. (2017). `CAclustering` applies `k-means` on the first K leading eigenvectors of $L_\tau + \alpha XX'$, where L_τ is the regularized graph Laplacian, X is the covariates matrix, and α is a tuning parameter.

Value

`estall`	A factor indicating nodes' labels. Items sharing the same label are in the same community.

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; p =5; prob1 = 0.9;
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)]
Q = 0.1*matrix(sign(runif(p*K) - 0.5), nrow = p);
for(i in 1:K){
  Q[(i-1)*(p/K)+(1:(p/K)), i] = 0.3; #remark. has a change here
}
W = matrix(0, nrow = n, ncol = K);
for(jj in 1:n) {
  pp = rep(1/(K-1), K); pp[l[jj]] = 0;
  if(runif(1) <= prob1) {W[jj, 1:K] = Pi[jj, ];}
  else
  W[jj, sample(K, 1, prob = pp)] = 1;
  }
W = t(W)
D0 = Q %*% W
D = matrix(0, n, p)
for (i in 1:n){
  D[i,] = rnorm(p, mean = D0[,i], sd = 1);
}
CAclustering(Adj, D, 2)
```

`Cov_based` *Covariates-based Spectral Clustering.*

Description

Covariates-based Spectral Clustering is a spectral clustering method that focuses solely on the covariates structure, i.e., the XX' where X is the covariates matrix, as introduced in Lee et al. (2010).

Usage

```
Cov_based(Covariate, K, itermax = 100, startn = 10)
```

Arguments

`Covariate`	An $n \times p$ covariate matrix. The rows correspond to nodes and the columns correspond to covariates.
`K`	A positive integer which is no larger than n. This is the predefined number of communities.
`itermax`	`k-means` parameter, indicating the maximum number of iterations allowed. The default value is 100.
`startn`	`k-means` parameter. The number of times the algorithm should be run with different initial centroids. The default value is 10.

Value

`estall`	A factor indicating nodes' labels. Items sharing the same label are in the same community.

References

Lee, A. B., Luca, D., Klei, L., Devlin, B., & Roeder, K. (2010). Discovering genetic ancestry using spectral graph theory. *Genetic Epidemiology: The Official Publication of the International Genetic Epidemiology Society*, 34(1), 51-59.
doi:10.1002/gepi.20434

Examples

```
# Simulate the Covariate Matrix
n = 10; p = 5; K = 2; prob1 = 0.9;
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
}
Q = 0.1*matrix(sign(runif(p*K) - 0.5), nrow = p);
for(i in 1:K){
  Q[(i-1)*(p/K)+(1:(p/K)), i] = 0.3; #remark. has a change here
}
W = matrix(0, nrow = n, ncol = K);
for(jj in 1:n) {
  pp = rep(1/(K-1), K); pp[l[jj]] = 0;
  if(runif(1) <= prob1) {W[jj, 1:K] = Pi[jj, ];}
  else
  W[jj, sample(K, 1, prob = pp)] = 1;
  }
W = t(W)
D0 = Q %*% W
D = matrix(0, n, p)
for (i in 1:n){
  D[i,] = rnorm(p, mean = D0[,i], sd = 1);
}
Cov_based(D, 2)
```

NAC *Spectral Clustering on Network-Adjusted Covariates.*

Description

Using network-adjusted covariates to detect underlying communities.

Usage

```
NAC(Adj, Covariate, K, alpha = NULL, beta = 0, itermax = 100, startn = 10)
```

Arguments

Adj	An $n \times n$ symmetric adjacency matrix with diagonals being 0 and positive entries being 1.
Covariate	An $n \times p$ covariate matrix. The rows correspond to nodes and the columns correspond to covariates.
K	A positive integer which is no larger than n. This is the predefined number of communities.
alpha	An optional numeric vector to tune the weight of covariate matrix. The default value is $\frac{\bar{d}/2}{d_i/\log n + 1}$, where d_i is the degree of node i and $\bar{d}$ is the average degree.
beta	An optional parameter used when the covariate matrix X is uninformative. By default, β is set as 0 assuming X carries meaningful information. Otherwise, users can manually specify a positive value to weigh network information.
itermax	k-means parameter, indicating the maximum number of iterations allowed. The default value is 100.

`startn`	`k-means` parameter. The number of times the algorithm should be run with different initial centroids. The default value is 10.

Details

Spectral Clustering Network-Adjusted Covariates (NAC) is fully established in *Network-Adjusted Covariates for Community Detection* of Hu & Wang (2023). This method is particularly effective in the analysis of multiscale networks with covariates, addressing the challenge of misspecification between networks and covariates. `NAC` relies on the construction of network-adjusted covariate vectors $y_i = \alpha_i x_i + \sum_{j:A_{ij}=1} x_j, i \in 1, \cdots, n$, where the first part has the nodal covariate information and the second part conveys network information. By constructing $Y = (y_1, \cdots, y_n)' = AX + D_\alpha X$ where A is the adjacency matrix, X is the covariate matrix, and D_α is the diagonal matrix with diagonals as $\alpha_1, \cdots, \alpha_n$, `NAC` applies `K-means` on the first K normalized left singular vectors, treating each row as a data point. A notable feature of `NAC` is its tuning-free nature, where node-specific coefficient α_i is computed given the i-th node's degree. `NAC` allows for user-specified α_i as well. A generalization with uninformative covariates is considered by adjusting parameter β. As long as the covariates do provide information, the specification of β can be ignored.

Value

`estall`	A factor indicating nodes' labels. Items sharing the same label are in the same community.

References

Hu, Y., & Wang, W. (2023). Network-Adjusted Covariates for Community Detection. *arXiv preprint arXiv:2306.15616*.
https://arxiv.org/abs/2306.15616

Examples

```
# Simulate the Network
n = 10; K = 2; p =5; prob1 = 0.9;
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)]
Q = 0.1*matrix(sign(runif(p*K) - 0.5), nrow = p);
for(i in 1:K){
  Q[(i-1)*(p/K)+(1:(p/K)), i] = 0.3; #remark. has a change here
}
W = matrix(0, nrow = n, ncol = K);
for(jj in 1:n) {
  pp = rep(1/(K-1), K); pp[l[jj]] = 0;
  if(runif(1) <= prob1) {W[jj, 1:K] = Pi[jj, ];}
  else
  W[jj, sample(K, 1, prob = pp)] = 1;
  }
W = t(W)
D0 = Q %*% W
D = matrix(0, n, p)
for (i in 1:n){
  D[i,] = rnorm(p, mean = D0[,i], sd = 1);
}
NAC(Adj, D, 2)
```

`Net_based` *Network-based Regularized Spectral Clustering.*

Description

Network-based Regularized Spectral Clustering is a spectral clustering with regularized Laplacian method, fully established in fully established in *Impact of Regularization on Spectral Clustering* of Joseph & Yu (2016).

Usage

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

Arguments

`Adj`	An $n \times n$ symmetric adjacency matrix with diagonals being 0 and positive entries being 1.
`K`	A positive positive integer which is no larger than n. This is the predefined number of communities.
`tau`	An optional tuning parameter to add J to the adjacency matrix A, where J is a constant matrix with all entries equal to $1/n$. The default value is the mean of nodes' degrees.
`itermax`	`k-means` parameter, indicating the maximum number of iterations allowed. The default value is 100.
`startn`	`k-means` parameter. The number of times the algorithm should be run with different initial centroids. The default value is 10.

Value

`estall`	A factor indicating nodes' labels. Items sharing the same label are in the same community.

References

Joseph, A., & Yu, B. (2016). Impact of Regularization on Spectral Clustering. *The Annals of Statistics*, 44(4), 1765-1791.
doi:10.1214/16AOS1447

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)
```

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

Description

Using ratios-of-eigenvectors to detect underlying communities.

Usage

```
SCORE(G, K, itermax = 100, startn = 10)
```

Arguments

G	An $n \times n$ symmetric adjacency matrix with diagonals being 0 and positive entries being 1, where isolated nodes are not allowed.
K	A positive integer which is no larger than n. This is the predefined number of communities.
itermax	k-means parameter, indicating the maximum number of iterations allowed. The default value is 100.
startn	k-means parameter. The number of times the algorithm should be run with different initial centroids. The default value is 10.

Details

SCORE is fully established in *Fast community detection by SCORE* of Jin (2015). SCORE uses the entrywise 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 factor indicating nodes' labels. Items sharing the same label are in the same community.

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)
```

SDP	*Semidefinite programming for Community Detection in Networks with Covariates.*

Description

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

Usage

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

Arguments

Adj	An $n \times n$ symmetric adjacency matrix with diagonals being 0 and positive entries being 1.
Covariate	An $n \times p$ 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 which is no larger than n. This is the predefined number of communities.
alpha	The element-wise upper bound in the SDP.
rho	The learning rate of SDP.
TT	The maximum of iteration.
tol	The tolerance for stopping criterion.
quiet	An optional input, indicating whether to print result at each step.
report_interval	An optional input. The frequency to print intermediate result.
r	An optional input. The expected rank of the solution, leave NULL if no constraint is required.

Details

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

Value

estall	A factor indicating nodes' labels. Items sharing the same label are in the same community.

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; p =5; prob1 = 0.9;
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)]
Q = 0.1*matrix(sign(runif(p*K) - 0.5), nrow = p);
for(i in 1:K){
  Q[(i-1)*(p/K)+(1:(p/K)), i] = 0.3; #remark. has a change here
}
W = matrix(0, nrow = n, ncol = K);
for(jj in 1:n) {
  pp = rep(1/(K-1), K); pp[l[jj]] = 0;
  if(runif(1) <= prob1) {W[jj, 1:K] = Pi[jj, ];}
  else
  W[jj, sample(K, 1, prob = pp)] = 1;
  }
W = t(W)
D0 = Q %*% W
D = matrix(0, n, p)
for (i in 1:n){
  D[i,] = rnorm(p, mean = D0[,i], sd = 1);
}
SDP(Adj, D, lambda = 0.2, K = 2, alpha = 0.5, rho = 2, TT = 100, tol = 5)
```

Index