Package 'TestBMN'

May 10, 2017

Type Package

Title Testing for binary Markov networks

Version 1.0

Date 2017-05-10

Author Jing Ma

Maintainer Jing Ma <crystal.jing.ma@gmail.com>

Description Conduct one-sample and two-sample testing of binary Markov networks. The test can be at the network level (global) or at the edge level (multiple).

Depends BMN, evd, glmnet, gdata, igraph, Matrix

License GPL (>=2)

LazyLoad yes

R topics documented:

`TestBMN-package` *Testing for binary Markov networks*

Description

The TestBMN-package provides a testing framework for detecting differential binary Markov networks in the case of two-sample testing, or one binary Markov network in the case of one-sample testing.

Details

The DESCRIPTION file: This package was not yet installed at build time.

Index: This package was not yet installed at build time.

Binary Markov networks are commonly used to model the conditional independence relationships between the components of a random vector. Specifically, let X be a p-dimensional random vector. The pairwise relationships in X can be captured by an undirected graph $G = (V, E)$, whose vertex set $V = \{1, \ldots, p\}$ and edge set E corresponds to conditional dependence. The binary Markov network associated with G has the joint distribution

$$p(X) \propto \exp \left\{ \sum_{(r,t)\in E} \theta_{r,t} X_r X_t \right\}$$

subject to a normalizing constant.

Given n_1 i.i.d. observations from one population and n_2 i.i.d. observations from another population, it is often of interest to test whether the two underlying Markov networks are the same. Let Θ_1 and Θ_2 be the two unknown partial correlation matrices from the two populations. It suffices to test

$$H_0 : \Theta_1 = \Theta_2 \text{ v.s. } H_1 : \Theta_1 \neq \Theta_2.$$

If the global null H_0 is rejected, it becomes of interest to test

$$H_{0,r,t} : \theta_{r,t,1} - \theta_{r,t,2} = 0, \text{ v.s. } H_{1,r,t} : \theta_{r,t,1} - \theta_{r,t,2} \neq 0, 1 \leq r < t \leq p$$

so as to recover the differential network $\Theta_1 - \Theta_2$.

The TestBMN-package provides a rigorous framework for the above two-sample global and multiple testing problems via `testTwoBMN`. The global testing procedure in `testTwoBMN` is particularly powerful against alternatives where the differential network $\Theta_1 - \Theta_2$ is sparse. The multiple testing procedure `testTwoBMN` provides asymptotic control of the false discovery proportion and the false discovery rate under suitable conditions. In addition, one can similarly conduct one-sample global and multiple testing problems via `testOneBMN`, if the goal is to estimate a binary Markov network with false discovery rate control.

More details about the method implemented in the TestBMN-package are available in Cai et al. (2017+).

Author(s)

Jing Ma <crystal.jing.ma@gmail.com>

References

Cai, T. T., Li, H., Ma, J. & Xia, Y. (2017+). A testing framework for detecting differential micorbial networks.

Xia, Y., Cai, T., & Cai, T. T. (2015). Testing differential networks with applications to the detection of gene-gene interactions. Biometrika, 102(2), 247-266.

Ravikumar, P., Wainwright, M. J., & Lafferty, J. D. (2010). High-dimensional Ising model selection using l1-regularized logistic regression. The Annals of Statistics, 38(3), 1287-1319.

Examples

```
library(igraph)
library(evd)
library(gdata)
library(glmnet)

set.seed(1)

p = 50    # number of variables
n = 100   # number of observations per replicate
n0 = 1000 # burn in tolerance
rho_high = 0.5  # signal strength
rho_low = 0.1   # signal strength
ncond = 2       # number of conditions to compare
eps = 8/n       # tolerance for extreme proportioned observations
q = (p*(p - 1))/2

##---(1) Generate the network
g_sf = sample_pa(p, directed=FALSE)
Amat = as.matrix(get.adjacency(g_sf, type="both"))

##---(2) Generate the Theta
Theta = vector("list", ncond)
weights = matrix(0, p, p)
upperTriangle(weights) = runif(q, rho_low, rho_high) * (2*rbinom(q, 1, 0.5) - 1)
weights = weights + t(weights)
Theta[[1]] = weights * Amat

##---(3) Generate the difference matrix
Delta = matrix(0, p, p)
upperTriangle(Delta) = runif(q, 1, 2)*sqrt(log(p)/n) *
                  (match(1:q, sample(q, q*0.1), nomatch = 0)>0)*(2*rbinom(q, 1, 0.5) - 1)
Delta = Delta + t(Delta)

Theta[[2]] = Theta[[1]] + Delta
Theta[[1]] = Theta[[1]] - Delta

##---(4) Simulate data and choose tuning parameter
dat = vector("list", ncond)
lambda = vector("list", ncond)
for (k in 1:ncond){
  dat[[k]] = BMN.samples(Theta[[k]], n, n0, skip=1)
  tmp = sapply(1:p, function(i) as.numeric(table(dat[[k]][,i]))[1]/n )
  while(min(tmp)<eps || abs(1-max(tmp)<eps)){
    dat[[k]] = BMN.samples(Theta[[k]], n, n0, skip=10)
    tmp = sapply(1:p, function(i) as.numeric(table(dat[[k]][,i]))[1]/n )
  }
}

tune = multiple.tune.twosample(dat, 1:20, verbose = TRUE)
for (k in 1:ncond){
  empcov <- cov(dat[[k]])
  lambda[[k]] = (tune$b/20) * sqrt( diag(empcov) * log(p)/n )
}

##---(5) Two-sample testing
```

```
test = testTwoBMN(dat, lambda, alpha.multi = 0.20)
```

`BMN.logistic` *Estimation of a binary Markov network using nodewise logistic regressions*

Description

This function aims to estimate the partial correlation matrix associated with a binary Markov network using the nodewise logistic regression approach proposed by Ravikumar et al. (2010).

Usage

```
BMN.logistic(X, lambda, gamma = 0.25, bic = TRUE, verbose = FALSE, eps = 1e-08)
```

Arguments

`X`	The $n \times p$ data matrix.
`lambda`	A vector of tuning parameters. The length of `lambda` should be p.
`gamma`	A tuning parameter required in evaluating the extended version of Bayesian information criterion (EBIC), which can be any constant between 0 and 1. Default is 0.25. See 'Details'.
`bic`	Whether to compute the EBIC. Default is TRUE.
`verbose`	Whether to print out intermediate iterations for every nodewise regression. Default is FALSE.
`eps`	Numeric scalar $>= 0$, indicating the tolerance level for differentiating zero and non-zero edges: entries $<$ `eps` will be set to 0.

Details

The function `BMN.logistic` fits p ℓ_1-regularized logistic regressions to the data X to recover the partial correlation matrix of the binary Markov network. Internally, the function `glmnet` is called for each node $j = 1, \ldots, p$ using the j-th column of data X as the response and all remaining variables as the predictors to estimate the neighborhood of node j. The j-th component of `lambda` is used as the penalization parameter for the j-th logisitc regression. Finally, the results from p regressions are aggregated to obtain the symmetric partial correlation matrix.

Model selection for each of the p regressions in `BMN.logistic` is done by minimizing the EBIC (Barber and Darton, 2015), where the additional parameter `gamma` corresponds to some prior belif on the set of considered models. For details, please refer to Barber and Darton (2015) and Zak-Szatkowska and Bogdan (2011).

Value

`theta`	The partial correlation matrix ($p \times p$) of the binary Markov network.
`adj`	The adjacency matrix ($p \times p$) of the binary Markov network.
`EBIC`	The extended version of Bayesian information criterion ($1 \times p$) for model selection.
`lambda`	The tuning parameter used ($1 \times p$).

Author(s)

Jing Ma

References

Ravikumar, P., Wainwright, M. J., & Lafferty, J. D. (2010). High-dimensional Ising model selection using l1-regularized logistic regression. The Annals of Statistics, 38(3), 1287-1319.

Zak-Szatkowska, M., & Bogdan, M. (2011). Modified versions of the Bayesian information criterion for sparse generalized linear models. Computational Statistics & Data Analysis, 55(11), 2908-2924.

Barber, R. F., & Drton, M. (2015). High-dimensional Ising model selection with Bayesian information criteria. Electronic Journal of Statistics, 9(1), 567-607.

See Also

`BMNPseudo`, `glmnet`

Examples

```
library(glmnet)
library(gdata)
library(igraph)

set.seed(1)

p = 50    # number of variables
n = 100   # number of observations per replicate
n0 = 1000 # burn in tolerance
rho_high = 0.5  # signal strength
rho_low = 0.1   # signal strength
eps = 8/n       # tolerance for extreme proportioned observations
q = (p*(p - 1))/2

##---(1) Generate the network
g_sf = sample_pa(p, directed=FALSE)
Amat = as.matrix(get.adjacency(g_sf, type="both"))

##---(2) Generate the Theta
weights = matrix(0, p, p)
upperTriangle(weights) = runif(q, rho_low, rho_high) * (2*rbinom(q, 1, 0.5) - 1)
weights = weights + t(weights)
Theta = weights * Amat
dat = BMN.samples(Theta, n, n0, skip=1)
tmp = sapply(1:p, function(i) as.numeric(table(dat[,i]))[1]/n )
while(min(tmp)<eps || abs(1-max(tmp)<eps)){
  dat = BMN.samples(Theta, n, n0, skip=10)
  tmp = sapply(1:p, function(i) as.numeric(table(dat[,i]))[1]/n )
}

lambda = rep(0.1, p)
fit = BMN.logistic(dat, lambda)
```

BMN.samples — *Sampling data using Gibbs sampling for use in examples*

Description

The function BMN.samples samples from a binary Markov network using Gibbs sampling. The resulting data matrix has binary measurements $\{-1, 1\}$.

Usage

```
BMN.samples(theta, numSamples, burnIn, skip)
```

Arguments

theta	The $p \times p$ partial correlation matrix corresponding to the binary Markov network.
numSamples	Number of samples to return.
burnIn	Number of samples to discard as burn in.
skip	Number of samples to discard in-btween returned samples.

Details

The function BMN.samples works similarly as BMNSamples in the BMN-package. The only difference between the two versions is that BMN.samples implemented here returns $\{-1, 1\}$ as the binary outcome, whereas the one in the BMN-package returns $\{-1, 1\}$ as the outcome.

Value

An $n \times p$ data matrix consisting of $\{-1, 1\}$.

Author(s)

Jing Ma

See Also

BMNSamples

Examples

```
library(gdata)
library(igraph)
set.seed(1)

p = 50    # number of variables
n = 100   # number of observations per replicate
n0 = 1000 # burn in tolerance
rho_high = 0.5  # signal strength
rho_low = 0.1   # signal strength
ncond = 2       # number of conditions to compare
eps = 8/n       # tolerance for extreme proportioned observations
q = (p*(p - 1))/2
```

```
##---(1) Generate the network
g_sf = sample_pa(p, directed=FALSE)
Amat = as.matrix(get.adjacency(g_sf, type="both"))

##---(2) Generate the Theta
weights = matrix(0, p, p)
upperTriangle(weights) = runif(q, rho_low, rho_high) * (2*rbinom(q, 1, 0.5) - 1)
weights = weights + t(weights)
Theta = weights * Amat
dat = BMN.samples(Theta, n, n0, skip=1)
tmp = sapply(1:p, function(i) as.numeric(table(dat[,i]))[1]/n )
while(min(tmp)<eps || abs(1-max(tmp)<eps)){
  dat = BMN.samples(Theta, n, n0, skip=10)
  tmp = sapply(1:p, function(i) as.numeric(table(dat[,i]))[1]/n )
}
```

global.tune *Model selection for global testing*

Description

The function global.tune selects the optimal tuning parameters for the global testing problem $H_0 : \Theta = 0$ or $H_0 : \Theta_1 = \Theta_2$.

Usage

```
global.tune(X, Lambda, gamma = 0.25)
```

Arguments

X	The $n \times p$ data matrix.
Lambda	The $nlambda \times p$ matrix of tuning parameters, each row representing one candidate vector of tuning parameter.
gamma	A control parameter used in evaluating the extended version of Bayesian information criterion (EBIC), which can be any constant between 0 and 1. Default is 0.25. See 'Details'.

Details

Model selection for global testing is done via the extended version of Bayesian information criterion (EBIC). The function global.tune should be called twice in two-sample testing problems such that the optimal tuning parameters for each population can be selected.

Value

EBIC	The extended version of Bayesian information criterion ($nlambda \times p$) for model selection.
lambdaOpt	The selected optimal tuning parameter ($1 \times p$).

Author(s)

Jing Ma

References

Barber, R. F., & Drton, M. (2015). High-dimensional Ising model selection with Bayesian information criteria. Electronic Journal of Statistics, 9(1), 567-607.

See Also

BMN.logistic

Examples

```
library(evd)
library(gdata)
library(glmnet)
library(igraph)

set.seed(1)

p = 50    # number of variables
n = 100   # number of observations per replicate
n0 = 1000 # burn in tolerance
rho_high = 0.5  # signal strength
rho_low = 0.1   # signal strength
ncond = 2       # number of conditions to compare
eps = 8/n       # tolerance for extreme proportioned observations
q = (p*(p - 1))/2

##---(1) Generate the network
g_sf = sample_pa(p, directed=FALSE)
Amat = as.matrix(get.adjacency(g_sf, type="both"))

##---(2) Generate the Theta
weights = matrix(0, p, p)
upperTriangle(weights) = runif(q, rho_low, rho_high) * (2*rbinom(q, 1, 0.5) - 1)
weights = weights + t(weights)
Theta = weights * Amat
dat = BMN.samples(Theta, n, n0, skip=1)
tmp = sapply(1:p, function(i) as.numeric(table(dat[,i]))[1]/n )
while(min(tmp)<eps || abs(1-max(tmp)<eps)){
  dat = BMN.samples(Theta, n, n0, skip=10)
  tmp = sapply(1:p, function(i) as.numeric(table(dat[,i]))[1]/n )
}
empcov = cov(dat)
LambdaMat = outer(seq(1,15), sqrt(0.01 * diag(empcov) * log(p)/n))
tune = global.tune(dat, LambdaMat)
lambda = tune$lambdaOpt
```

---

multiple.tune.onesample *Model selection for one-sample multiple testing*

Description

The function multiple.tune.onesample selects the optimal tuning parameters for the multiple testing problem $H_0 : \theta_{r,t} = 0$ for all $1 \le r < t \le p$.

Usage

```
multiple.tune.onesample(X, B, verbose = FALSE)
```

Arguments

`X`	The $n \times p$ data matrix.
`B`	The set of candidate integer multipliers.
`verbose`	Whether to print out intermediate iteration steps. Default is `FALSE`.

Details

The false discovery rate control in the multiple testing problem

$$H_0 : \theta_{r,t} = 0, 1 \leq r < t \leq p$$

is based on approximating the number of false discoveries by $2 - 2\Phi(t) * p * (p-1)/2$. Thus the optimal tuning parameter is selected with the principle of making the above approximation error to be as small as possible. The candidate tuning parameters are selected using a data-driven approach, therefore the user only needs to specify a candidate set of integer multipliers. Details on how the approximation error is defined are available in Xia et al. (2014).

Value

`error`	The empirical version of the approximation error, evaluated over the range of integer multipliers in `B`.
`b`	The optimal integer multiplier.

Author(s)

Jing Ma

References

Xia, Y., Cai, T., & Cai, T. T. (2015). Testing differential networks with applications to the detection of gene-gene interactions. Biometrika, 102(2), 247-266.

See Also

`testOneBMN`

Examples

```
library(evd)
library(gdata)
library(glmnet)
library(igraph)

set.seed(1)

p = 50    # number of variables
n = 100   # number of observations per replicate
n0 = 1000 # burn in tolerance
rho_high = 0.5  # signal strength
rho_low = 0.1   # signal strength
```

```
ncond = 2       # number of conditions to compare
eps = 8/n       # tolerance for extreme proportioned observations
q = (p*(p - 1))/2

##---(1) Generate the network
g_sf = sample_pa(p, directed=FALSE)
Amat = as.matrix(get.adjacency(g_sf, type="both"))

##---(2) Generate the Theta
weights = matrix(0, p, p)
upperTriangle(weights) = runif(q, rho_low, rho_high) * (2*rbinom(q, 1, 0.5) - 1)
weights = weights + t(weights)
Theta = weights * Amat
dat = BMN.samples(Theta, n, n0, skip=1)
tmp = sapply(1:p, function(i) as.numeric(table(dat[,i]))[1]/n )
while(min(tmp)<eps || abs(1-max(tmp)<eps)){
  dat = BMN.samples(Theta, n, n0, skip=10)
  tmp = sapply(1:p, function(i) as.numeric(table(dat[,i]))[1]/n )
}

tune = multiple.tune.onesample(dat, 1:20, verbose = TRUE)
```

multiple.tune.twosample

Model selection for two-sample multiple testing

Description

The function `multiple.tune.twosample` selects the optimal tuning parameters for the multiple testing problem $H_0 : \theta_{r,t,1} - \theta_{r,t,2} = 0$ for all $1 \leq r < t \leq p$.

Usage

```
multiple.tune.twosample(dat, B, verbose = FALSE)
```

Arguments

`dat`	A list of length 2, consisting of the two data matrices generated from two populations. The data matrix from the k-th population should be of dimension $n_k \times p$ for $k = 1, 2$.
`B`	The set of candidate integer multipliers.
`verbose`	Whether to print out intermediate iteration steps. Default is `FALSE`.

Details

The false discovery rate control in the multiple testing problem

$$H_0 : \theta_{r,t,1} - \theta_{r,t,2} = 0, 1 \leq r < t \leq p$$

is based on approximating the number of false discoveries by $2 - 2\Phi(t) * p * (p - 1)/2$. Thus the optimal tuning parameter is selected with the principle of making the above approximation error to be as small as possible. The candidate tuning parameters are selected using a data-driven approach, therefore the user only needs to specify a candidate set of integer multipliers. Details on how the approximation error is defined are available in Xia et al. (2014).

Value

error	The empirical version of the approximation error, evaluated over the range of integer multipliers in B.
b	The optimal integer multiplier.

Author(s)

Jing Ma

References

Xia, Y., Cai, T., & Cai, T. T. (2015). Testing differential networks with applications to the detection of gene-gene interactions. Biometrika, 102(2), 247-266.

See Also

testTwoBMN, multiple.tune.onesample

Examples

```
library(igraph)
library(evd)
library(gdata)
library(glmnet)

set.seed(1)

p = 50    # number of variables
n = 100   # number of observations per replicate
n0 = 1000 # burn in tolerance
rho_high = 0.5  # signal strength
rho_low = 0.1   # signal strength
ncond = 2       # number of conditions to compare
eps = 8/n       # tolerance for extreme proportioned observations
q = (p*(p - 1))/2

##---(1) Generate the network
g_sf = sample_pa(p, directed=FALSE)
Amat = as.matrix(get.adjacency(g_sf, type="both"))

##---(2) Generate the Theta
Theta = vector("list", ncond)
weights = matrix(0, p, p)
upperTriangle(weights) <- runif(q, rho_low, rho_high) * (2*rbinom(q, 1, 0.5) - 1)
weights = weights + t(weights)
Theta[[1]] = weights * Amat

##---(3) Generate the difference matrix
Delta <- matrix(0, p, p)
upperTriangle(Delta) <- runif(q, 1, 2) * sqrt(log(p)/n) *
                 (match(1:q, sample(q, q*0.1), nomatch = 0)>0)*(2*rbinom(q, 1, 0.5) - 1)
Delta <- Delta + t(Delta)

Theta[[2]] = Theta[[1]] + Delta
Theta[[1]] = Theta[[1]] - Delta
```

```
##---(4) Simulate data and choose tuning parameter
dat = vector("list", ncond)
lambda = vector("list", ncond)
for (k in 1:ncond){
  dat[[k]] = BMN.samples(Theta[[k]], n, n0, skip=1)
  tmp = sapply(1:p, function(i) as.numeric(table(dat[[k]][,i]))[1]/n )
  while(min(tmp)<eps || abs(1-max(tmp)<eps)){
    dat[[k]] = BMN.samples(Theta[[k]], n, n0, skip=10)
    tmp = sapply(1:p, function(i) as.numeric(table(dat[[k]][,i]))[1]/n )
  }
}

tune = multiple.tune.twosample(dat, 1:20, verbose = TRUE)
for (k in 1:ncond){
  empcov <- cov(dat[[k]])
  lambda[[k]] = (tune$b/20) * sqrt( diag(empcov) * log(p)/n )
}

##---(5) Two-sample testing
test = testTwoBMN(dat, lambda, alpha.multi = 0.20)
```

testOneBMN — *One-sample testing of a binary Markov network*

Description

testOneBMN tests whether a binary Markov network is empty, i.e. whether the corresponding partial correlation matrix $\Theta = 0$. In addition, testOneBMN simultaneously tests whether $\theta_{rt} = 0$ for each pair of edge (r, t) to recover the Markov network with false discovery rate control.

Usage

```
testOneBMN(X, lambda, alpha.global = 0.05, multiTest = TRUE, alpha.multi = 0.1)
```

Arguments

X	The $n \times p$ data matrix.
lambda	The $1 \times p$ vector of tuning parameters to be used in BMN.logistic for estimating an initial partial correlation matrix.
alpha.global	The significance level for global testing. Default is 0.05.
multiTest	Whether to conduct multiple testing of pairwise edges. Default is TRUE.
alpha.multi	The level of false discovery rate in multiple testing, necessary if multiTest=TRUE. Default is 0.10.

Details

The function `testOneBMN` implements the one-sample testing of a binary Markov network. It calculates the standardized pairwise statistic $W_{r,t}$ for each pair of edge (r,t) using the data `X` together with the tuning parameter `lambda`. The test statistic for the global test $\Theta = 0$ is

$$M_{n,p} = \max_{1\le r<t\le p} W_{r,t}^2$$

The null $\Theta = 0$ is rejected if

$$M_{n,p} - 4\log(p) + \log(\log(p)) \ge q_\alpha$$

where q_α is the $1-\alpha$ quantile of the type I extreme value distribution with cumulative distribution function $\exp -(8\pi)^{-1/2}e^{-z/2}$. To recover the underlying Markov network, `testOneBMN` considers the multiple testing problem

$$H_{0,r,t} : \theta_{r,t} = 0, 1 \le r < t \le p$$

The test statistic for each individual hypothesis $H_{0,r,t}$ is $W_{r,t}$. The multiple testing procedure in `testOneBMN` rejects the null $H_{0,r,t}$ if $|W_{r,t}| > \tau$, where the threshold τ is carefully chosen to ensure false discovery rate control at the pre-specified level.

Note the tuning parameter `lambda` should be carefully chosen to ensure that the initial estimate of the partial correlation matrix is reasonably good. In particular, the optimal tuning parameters required in global testing and multiple testing may not be the same. See `global.tune` and `multiple.tune.onesample` for how to choose `lambda`. More details on the testing procedures are available in Cai et al. (2017+).

`testOneBMN` calls `BMN.logistic` to estimate an initial partial correlation matrix using nodewise ℓ_1-regularized logistic regressions. As a result, `testOneBMN` returns a warning message if "one multinomial or binomial class has 1 or 0 observations" occurs in any of the p logisitc regressions. The user should double check the data matrix `X` to avoid such situations before applying `testOneBMN` again.

Value

`statistic`	The test statistic.
`pvalue`	The p-value for testing the null $\Theta = 0$.
`reject`	Whether to reject the null $\Theta = 0$.
`W`	A $p \times p$ matrix consisting of the standardized pairwise statistics.
`thetaInit`	The initial estimated partial correlation matrix using `BMN.logistic`.
`theta`	The de-sparsified partial correlation matrix.
`network`	The estimated network with false discovery rate control at the pre-specified level `alpha.multi`.

Author(s)

Jing Ma

References

Cai, T. T., Li, H., Ma, J. & Xia, Y. (2017+). A testing framework for detecting differential micorbial networks.

See Also

testTwoBMN, global.tune, multiple.tune.onesample, BMN.logistic

Examples

```
library(glmnet)
library(gdata)
library(evd)

set.seed(1)

p = 50    # number of variables
n = 100   # number of observations per replicate
n0 = 1000 # burn in tolerance
rho_high = 0.5  # signal strength
rho_low = 0.1   # signal strength
eps = 8/n       # tolerance for extreme proportioned observations
q = (p*(p - 1))/2

##---(1) Generate the network
g_sf = sample_pa(p, directed=FALSE)
Amat = as.matrix(get.adjacency(g_sf, type="both"))

##---(2) Generate the Theta
weights = matrix(0, p, p)
upperTriangle(weights) = runif(q, rho_low, rho_high) * (2*rbinom(q, 1, 0.5) - 1)
weights = weights + t(weights)
Theta = weights * Amat
dat = BMN.samples(Theta, n, n0, skip=1)
tmp = sapply(1:p, function(i) as.numeric(table(dat[,i]))[1]/n )
while(min(tmp)<eps || abs(1-max(tmp)<eps)){
  dat = BMN.samples(Theta, n, n0, skip=10)
  tmp = sapply(1:p, function(i) as.numeric(table(dat[,i]))[1]/n )
}

lambda = rep(0.05, p)
test1 = testOneBMN(dat, lambda)
```

---

testTwoBMN *Two-sample testing of binary Markov networks*

Description

testTwoBMN tests whether two binary Markov networks are the same, i.e. whether the corresponding partial correlation matrices $\Theta_1 = \Theta_2$.

Usage

```
testTwoBMN(dat, lambda, alpha.global = 0.05, multiTest = TRUE, alpha.multi = 0.1)
```

Arguments

`dat`	A list of length 2, consisting of the two data matrices generated from two populations. The data matrix from the k-th population should be of dimension $n_k \times p$ for $k = 1, 2$.
`lambda`	A list of length 2, consisting of the two $1 \times p$ vectors of tuning parameters to be used in `BMN.logistic` for estimating the initial partial correlation matrix in each population.
`alpha.global`	The significance level for global testing. Default is 0.05.
`multiTest`	Whether to conduct multiple testing of pairwise edges. Default is `TRUE`.
`alpha.multi`	The level of false discovery rate in multiple testing, necessary if `multiTest=TRUE`. Default is 0.10.

Details

The function `testTwoBMN` implements the two-sample testing of binary Markov networks. It calculates the standardized pairwise statistic $W_{r,t}$ for each pair of edge (r, t) using the data matrices in `dat` together with the corresponding tuning parameters `lambda`. The test statistic for $\Theta_1 = \Theta_2$ is

$$M_{n,p} = \max_{1 \le r < t \le p} W_{r,t}^2$$

The null $\Theta_1 = \Theta_2$ is rejected if

$$M_{n,p} - 4\log(p) + \log(\log(p)) \ge q_\alpha$$

where q_α is the $1 - \alpha$ quantile of the type I extreme value distribution with cumulative distribution function $\exp -(8\pi)^{-1/2} e^{-z/2}$. To recover the differential Markov network, `testTwoBMN` considers the multiple testing problem

$$H_{0,r,t} : \theta_{r,t,1} - \theta_{r,t,2} = 0, 1 \le r < t \le p$$

The test statistic for each individual hypothesis $H_{0,r,t}$ is $W_{r,t}$. The multiple testing procedure in `testOneBMN` rejects the null $H_{0,r,t}$ if $|W_{r,t}| > \tau$, where the threshold τ is carefully chosen to ensure false discovery rate control at the pre-specified level.

Note the tuning parameters `lambda` should be carefully chosen to ensure that the initial estimate of the partial correlation matrix is reasonably good. In particular, the optimal tuning parameters required in global testing and multiple testing may not be the same. See `global.tune` and `multiple.tune.twosample` for how to choose `lambda`. More details on the testing procedures are available in Cai et al. (2017+).

`testTwoBMN` calls `BMN.logistic` to estimate the initial partial correlation matrices using nodewise l1-regularized logistic regressions before calling `debias`. As a result, `testTwoBMN` returns a warning message if "one multinomial or binomial class has 1 or 0 observations" occurs in any of the p logisitc regressions. The user should double check the data matrices in `dat` to avoid such situations before applying `testTwoBMN` again.

Value

`statistic`	The test statistic.
`pvalue`	The p-value for testing the null $\Theta_1 = \Theta_2$.
`reject`	Whether to reject the null $\Theta_1 = \Theta_2$.
`W`	A $p \times p$ matrix consisting of the standardized pairwise statistics.

`thetaXInit`	The initial estimated partial correlation matrix based on `X` using `BMN.logistic`.
`thetaX`	The de-sparsified partial correlation matrix for the first population.
`thetaYInit`	The initial estimated partial correlation matrix based on `Y` using `BMN.logistic`.
`thetaY`	The de-sparsified partial correlation matrix for the second population.
`network`	The estimated differential network with false discovery rate control at the pre-specified level `alpha.multi`.

Author(s)

Jing Ma

References

Cai, T. T., Li, H., Ma, J. & Xia, Y. (2017+). A testing framework for detecting differential micorbial networks.

See Also

`testOneBMN`, `global.tune`, `multiple.tune.twosample`, `BMN.logistic`

Examples

```
library(evd)
library(gdata)
library(glmnet)

set.seed(1)

p = 50    # number of variables
n = 100   # number of observations per replicate
n0 = 1000 # burn in tolerance
rho_high = 0.5  # signal strength
rho_low = 0.1   # signal strength
ncond = 2       # number of conditions to compare
eps = 8/n       # tolerance for extreme proportioned observations
q = (p*(p - 1))/2

##---(1) Generate the network
g_sf = sample_pa(p, directed=FALSE)
Amat = as.matrix(get.adjacency(g_sf, type="both"))

##---(2) Generate the Theta
Theta = vector("list", ncond)
weights = matrix(0, p, p)
upperTriangle(weights) <- runif(q, rho_low, rho_high) * (2*rbinom(q, 1, 0.5) - 1)
weights = weights + t(weights)
Theta[[1]] = weights * Amat

##---(3) Generate the difference matrix
Delta = matrix(0, p, p)
upperTriangle(Delta) <- runif(q, 1, 2)*sqrt(log(p)/n) *
                      (match(1:q, sample(q, 5), nomatch = 0)>0)*(2*rbinom(q, 1, 0.5) - 1)
Delta = Delta + t(Delta)

Theta[[2]] = Theta[[1]] + Delta
```

```
Theta[[1]] = Theta[[1]] - Delta

##---(4) Simulate data and choose tuning parameter
dat = vector("list", ncond)
lambda = vector("list", ncond)
for (k in 1:ncond){
  dat[[k]] = BMN.samples(Theta[[k]], n, n0, skip=1)
  tmp = sapply(1:p, function(i) as.numeric(table(dat[[k]][,i]))[1]/n )
  while(min(tmp)<eps || abs(1-max(tmp)<eps)){
    dat[[k]] = BMN.samples(Theta[[k]], n, n0, skip=10)
    tmp = sapply(1:p, function(i) as.numeric(table(dat[[k]][,i]))[1]/n )
  }
  empcov = cov(dat[[k]])
  LambdaMat = outer(seq(1,15), sqrt(0.01 * diag(empcov) * log(p)/n))
  tune = global.tune(dat[[k]], LambdaMat)
  lambda[[k]] = tune$lambdaOpt
}

##---(5) Two-sample testing
test = testTwoBMN(dat, lambda)
```

Index