Package: TransTGGM (via r-universe)

August 21, 2024

Title Transfer Learning for Tensor Graphical Models

Version 1.0.0

Description Tensor Gaussian graphical models (GGMs) have important applications in numerous areas, which can interpret conditional independence structures within tensor data. Yet, the available tensor data in one single study is often limited due to high acquisition costs. Although relevant studies can provide additional data, it remains an open question how to pool such heterogeneous data. This package implements a transfer learning framework for tensor GGMs, which takes full advantage of informative auxiliary domains even when non-informative auxiliary domains are present, benefiting from the carefully designed data-adaptive weights. Reference: Ren, M., Zhen Y., and Wang J. (2022). ``Transfer learning for tensor graphical models" <arXiv:2211.09391>.

License GPL-2

Encoding UTF-8

Imports MASS, Matrix, rTensor, Tlasso, glasso, doParallel,expm

LazyData true

LazyLoad yes

LazyDataCompression xz

RoxygenNote 7.1.2

Depends R (>= 3.5.0)

Suggests knitr, rmarkdown

VignetteBuilder knitr, rmarkdown

NeedsCompilation no

Author Mingyang Ren [aut, cre] (<https://orcid.org/0000-0002-8061-9940>), Yaoming Zhen [aut], Junhui Wang [aut]

Maintainer Mingyang Ren <renmingyang17@mails.ucas.ac.cn>

Repository CRAN

Date/Publication 2022-11-23 11:30:09 UTC

Contents

`example.data` *Some example data*

Description

Some example data

Format

A list including: t.data: Tensor data in the target domain with n=50 and (p1,p2,p3)=(10,10,10). A.data: Tensor data in auxiliary domains with K=5 and nk=100. t.Omega.true.list: A list, the true precision matrices of all modes in the target domain.

Source

Simulated data

`tensor.GGM.trans` *Transfer learning for tensor graphical models.*

Description

The main function for Transfer learning for tensor graphical models.

Usage

```
tensor.GGM.trans(t.data, A.data, A.lambda, A.orac = NULL, c=0.6,
                        t.lambda.int.trans=NULL, t.lambda.int.aggr=NULL,
                        theta.algm="cd", cov.select="inverse",
                        cov.select.agg.size = "inverse",
                        cov.select.agg.diff = "tensor.prod",
                        symmetric = TRUE, init.method="Tlasso",
                        init.method.aux="Tlasso", mode.set = NULL,
                        init.iter.Tlasso=2, cn.lam2=seq(0.1,2,length.out =10),
                        c.lam.Tlasso=20, c.lam.sepa=20, adjust.BIC=FALSE,
                        normalize = TRUE, inti.the=TRUE, sel.ind="fit")
```

Arguments

t.data	The tensor data in the target domain, a p1 * p2 * ... * pM * n array, where n is the sample size and pm is dimension of the m-th tensor mode. M should be larger than 2.
A.data	The tensor data in auxiliary domains, a list with K elements, each of which is a p1 * p2 * ... * pM * nk array, where nk is the sample size of the k-th auxiliary domain.
A.lambda	The tuning parameters used for initialization in auxiliary domains, a list with K elements, each of which is a M-dimensional vector corresponding to M modes.
A.orac	The set of informative auxiliary domains, and the default setting is NULL, which means that no set is specified.
c	The c of subjects in the target domain are used for initialization of the transfer learning, and the remaining 1-c of subjects are used for the model selection step. The default setting is 0.8.
t.lambda.int.trans	The tuning parameters used for initialization in the target domain (based on c subjects used for transfer learning), that is, the tuning lambda for Tlasso (PAMI, 2020) & Separable method (JCGS, 2022)
t.lambda.int.aggr	The tuning parameters used for initialization in the target domain (based on 1-c subjects used for the model selection step).
theta.algm	The optimization algorithm used to solve $\widehat{\Omega}$ in step 2(b), which can be selected as "admm" (ADMM algorithm) or "cd" (coordinate descent).
cov.select	Methods used to calculate covariance matrices for initialization in both target and auxiliary domains, which can be selected as "tensor.prod" (tensor product based on tensor subject and the initial estimate of the precision matrix, TPAMI, 2020) and "inverse" (direct inversion of the initial estimate of the precision matrix)
cov.select.agg.size	Methods used to calculate covariance matrices for model selection step in the target domain.
cov.select.agg.diff	Methods used to calculate covariance matrices for model selection step in the target domain.
symmetric	Whether to symmetrize the final estimated precision matrices, and the default is True.
init.method	The initialization method for tensor precision matrices in the target domain, which can be selected as "Tlasso" (PAMI, 2020) & "sepa" (Separable method, JCGS, 2022). Note that the "sepa" method has not been included in the current version of this R package to circumvent code ownership issues.
init.method.aux	The initialization method for tensor precision matrices in auxiliary domains.
mode.set	Whether to estimate only the specified mode, and the default setting is NULL, which means estimating all mode.

`init.iter.Tlasso` The number of maximal iteration when using Tlasso for initialization, default is 2.

`cn.lam2` The coefficient set in tuning parameters used to solve $\widehat{\Omega}$ in step 2(b), default is seq(0.1,1,length.out =10).

`c.lam.Tlasso` The coefficient in tuning parameters for initialization (when using Tlasso): $c.lam.Tlasso*\sqrt{(pm*\log(pm)/(n*p1*...*pM))}$, default is 20 suggested in (PAMI, 2020).

`c.lam.sepa` The coefficient in tuning parameters for initialization (when using sepa): $c.lam.sepa*\sqrt{(pm*\log(pm)/(n*p1*...*pM))}$.

`adjust.BIC` Whether to use the adjusted BIC to select lambda2, the default setting is F.

`normalize` The normalization method of precision matrix. When using Tlasso, $\Omega_{11}=1$ if normalize = F and $\|\Omega_{11}\|_F=1$ if normalize = T. Default value is T.

`inti.the` T: the initial values in Step 2(b) is Omega0.

`sel.ind` The approach to model selection, which can be selected from c("fit", "predict").

Value

A result list including: Omega.list: the final estimation result of the target precision matrices after the model selection of transfer learning-based estimation and initial estimation (in which the initial covariance matrices of auxiliary domains is weighted by sample sizes); Omega.sym.list: the symmetrized final estimation result in Omega.list; Omega.list.diff: the final estimation result of the target precision matrices after the model selection of transfer learning-based estimation and initial estimation (in which the initial covariance matrices of auxiliary domains is weighted by the differences with the target domain); Omega.sym.list.diff: the symmetrized final estimation result in Omega.list.diff; res.trans.list: transfer learning-based estimation results.

Author(s)

Mingyang Ren, Yaoming Zhen, Junhui Wang. Maintainer: Mingyang Ren renmingyang17@mails.ucas.ac.cn.

References

Ren, M., Zhen Y., and Wang J. (2022). Transfer learning for tensor graphical models.

Examples

```
library(TransTGGM)
library(Tlasso)
data(example.data)
t.data = example.data$t.data
A.data = example.data$A.data
t.Omega.true.list = example.data$t.Omega.true.list
normalize = TRUE

K = length(A.data)
p.vec = dim(t.data)
M = length(p.vec) - 1
n = p.vec[M+1]
```

```
p.vec = p.vec[1:M]
tla.lambda = 20*sqrt( p.vec*log(p.vec) / ( n * prod(p.vec) ))
A.lambda = list()
for (k in 1:K) {
  A.lambda[[k]] = 20*sqrt( log(p.vec) / ( dim(A.data[[k]])[M+1] * prod(p.vec) ))
}

res.final = tensor.GGM.trans(t.data, A.data, A.lambda, normalize = normalize)
Tlasso.Omega.list = Tlasso.fit(t.data, lambda.vec = tla.lambda,
                    norm.type = 1+as.numeric(normalize))

i.Omega = as.data.frame(t(unlist(est.analysis(res.final$Omega.list, t.Omega.true.list))))
i.Omega.diff = t(unlist(est.analysis(res.final$Omega.list.diff, t.Omega.true.list)))
i.Omega.diff = as.data.frame(i.Omega.diff)
i.Tlasso = as.data.frame(t(unlist(est.analysis(Tlasso.Omega.list, t.Omega.true.list))))
i.Omega.diff     # proposed.v
i.Omega          # proposed
i.Tlasso         # Tlasso
```

`Theta.est` *Fast sparse precision matrix estimation.*

Description

The fast sparse precision matrix estimation in step 2(b).

Usage

```
Theta.est(S.hat.A, delta.hat, lam2=0.1, Omega.hat0=NULL,
                n=100, max_iter=10, eps=1e-3, method = "cd")
```

Arguments

`S.hat.A`	The sample covariance matrix.
`delta.hat`	The divergence matrix estimated in step 2(a). If the precision matrix is estimated in the common case (Liu and Luo, 2015, JMVA), it can be set to zero matrix.
`lam2`	A float value, a tuning parameter.
`Omega.hat0`	The initial values of the precision matrix, which can be unspecified.
`n`	The sample size.
`max_iter`	Int, maximum number of cycles of the algorithm.
`eps`	A float value, algorithm termination threshold.
`method`	The optimization algorithm, which can be selected as "admm" (ADMM algorithm) or "cd" (coordinate descent).

Value

A result list including: Theta.hat.m: the optimal precision matrix; BIC.summary: the summary of BICs; Theta.hat.list.m: the precision matrices corresponding to a sequence of tuning parameters.

Author(s)

Mingyang Ren, Yaoming Zhen, Junhui Wang. Maintainer: Mingyang Ren renmingyang17@mails.ucas.ac.cn.

References

Ren, M., Zhen Y., and Wang J. (2022). Transfer learning for tensor graphical models. Liu, W. and Luo X. (2015). Fast and adaptive sparse precision matrix estimation in high dimensions, Journal of Multivariate Analysis.

Examples

```
p = 20
n = 200
omega = diag(rep(1,p))
for (i in 1:p) {
  for (j in 1:p) {
    omega[i,j] = 0.3^(abs(i-j))*(abs(i-j) < 2)
  }
}
Sigma = solve(omega)
X = MASS::mvrnorm(n, rep(0,p), Sigma)
S.hat.A = cov(X)
delta.hat = diag(rep(1,p)) - diag(rep(1,p))
omega.hat = Theta.est(S.hat.A, delta.hat, lam2=0.2)
```

`Theta.tuning` *Fast sparse precision matrix estimation.*

Description

The fast sparse precision matrix estimation in step 2(b).

Usage

```
Theta.tuning(lambda2, S.hat.A, delta.hat, Omega.hat0, n.A,
                    theta.algm="cd", adjust.BIC=FALSE)
```

Arguments

lambda2	A vector, a sequence of tuning parameters.
S.hat.A	The sample covariance matrix.
delta.hat	The divergence matrix estimated in step 2(a). If the precision matrix is estimated in the common case (Liu and Luo, 2015, JMVA), it can be set to zero matrix.
Omega.hat0	The initial values of the precision matrix.
n.A	The sample size.
theta.algm	The optimization algorithm used to solve $\widehat{\Omega}$ in step 2(b), which can be selected as "admm" (ADMM algorithm) or "cd" (coordinate descent).
adjust.BIC	Whether to use the adjusted BIC to select lambda2, the default setting is F.

Value

A result list including: Theta.hat.m: the optimal precision matrix; BIC.summary: the summary of BICs; Theta.hat.list.m: the precision matrices corresponding to a sequence of tuning parameters.

Author(s)

Mingyang Ren, Yaoming Zhen, Junhui Wang. Maintainer: Mingyang Ren renmingyang17@mails.ucas.ac.cn.

References

Ren, M., Zhen Y., and Wang J. (2022). Transfer learning for tensor graphical models. Liu, W. and Luo X. (2015). Fast and adaptive sparse precision matrix estimation in high dimensions, Journal of Multivariate Analysis.

Examples

```
p = 20
n = 200
omega = diag(rep(1,p))
for (i in 1:p) {
  for (j in 1:p) {
    omega[i,j] = 0.3^(abs(i-j))*(abs(i-j) < 2)
  }
}
Sigma = solve(omega)
X = MASS::mvrnorm(n, rep(0,p), Sigma)
S.hat.A = cov(X)
delta.hat = diag(rep(1,p)) - diag(rep(1,p))
lambda2 = seq(0.1,0.5,length.out =10)
res = Theta.tuning(lambda2, S.hat.A, delta.hat, n.A=n)
omega.hat = res$Theta.hat.m
```

Index