Package ‘pCIA’

March 5, 2018

Title Penalized Co-Inertia Analysis

Version 0.9

Author Eun Jeong Min, Sandra Safo, and Qi Long

Maintainer Eun Jeong Min <mineunj@pennmedicine.upenn.edu>

Description This package conduct penalized co-inertia analysis and generating output plot. Sparse co-inertia analysis and structured sparse co-inertia analysis models are implemented. Cross validation is also available for selecting optimal tuning parameters in each model. Output plot is also generated, which is originally provided by packages `ade4` and `made4` by changing input argument to enhance the readability of labels in the plot.

Depends R (>= 3.4.3), scatterplot3d, glmnet, ade4

License Artistic-2.0

Encoding UTF-8

LazyData true

RoxygenNote 6.0.1

Imports Matrix, made4

Suggests knitr, rmarkdown

VignetteBuilder knitr

R topics documented:

demoData ................................. 2
fitscia .................................. 3
fitssscia ................................ 4
genpInput ................................ 5
getNetMatrix ............................ 6
NCI60path ................................ 7
oCIA ...................................... 7
plot.pcia ................................ 8
sCIA.cv .................................. 9
ssCIA.cv ................................. 10

Index 12
**Description**

Reduced data used for the first scenario simulations in the paper.

**Usage**

```
data(demodata)
```

**Details**

- \( p = 400 \). dimension of features in the data \( X \)
- \( q = 500 \). dimension of features in the data \( Y \)
- \( n = 200 \). sample size for one simulated data.
- \( \text{nmcsample} = 10 \). number of Monte Carlo simulated data.
- \( K = 1 \). number of loading pairs to be estimated.
- \( \text{setX} \). (n*\text{nmcsample})-by-p data matrix
- \( \text{setY} \). (n*\text{nmcsample})-by-q data matrix
- \( D \). n-by-n diagonal weight matrix for samples.
- \( Qx \). p-by-p diagonal weight matrix for variables of \( X \).
- \( Qy \). q-by-q diagonal weight matrix for variables of \( Y \).
- \( ta \). true loading vector used to generate MC \( X \) datasets.
- \( tb \). true loading vector used to generate MC \( Y \) datasets.
- \( eX \). matrix that has two columns. Each rows contains location of variabes that are connected to each other in the data \( X \).
- \( eY \). matrix that has two columns. Each rows contains location of variabes that are connected to each other in the data \( Y \).

**References**

Penalized Co-Inertia Analysis with application to -Ominics data, Min et al. (2018)

**Examples**

```
data(demodata)
```
Sparse Co-Inercia Analysis

Description

This function conducts sparse CIA for a given tuning parameter value.

Usage

fitscia(X, Y, na, nb, lambdas, maxiter = 100)

Arguments

- X: n-by-p transformed data matrix of X, samples are rows and columns are features.
- Y: n-by-q transformed data matrix of Y, samples are rows and columns are features.
- na: p-by-1 vector, initial values of loading vector of variables in X.
- nb: q-by-1 vector, initial values of loading vector of variables in Y.
- lambdas: 2-by-1 vector, containing sparsity penalty parameter values for the features space of X and Y each.

Value

This function returns a list object.

- call: the matched call.
- res.opta: the estimated loading vector of transformed matrix of X.
- res.optb: the estimated loading vector of transformed matrix of Y.

See Also

see also oCIA, sCIA.cv.

Examples

```
library(pCIA)
data("demoData")
attach(demoData)
X <- sweep(X[1:n,], 2, apply(X, 2, mean))
Y <- sweep(Y[1:n,], 2, apply(Y, 2, mean))
result.cia <- oCIA(nX, nY, D, qx, Qy)
result.scia <- fitscia(nX, nY, result.cia$res.opta[,1], result.cia$res.optb[,1], c(6, 6))
```
Description

This function conducts structured sparse CIA for a given tuning parameter value.

Usage

fitsscia(X, Y, na, nb, Sx, Sy, lambdas, maxiter = 100)

Arguments

- X: n-by-p transformed data matrix of X, samples are rows and columns are features.
- Y: n-by-q transformed data matrix of Y, samples are rows and columns are features.
- na: p-by-1 vector, initial values of loading vector of variables in X.
- nb: q-by-1 vector, initial values of loading vector of variables in Y.
- Sx: \(Q_x^{-1/2}E_x\Gamma_x^{1/2}\) where \(E_x\) and \(\Gamma_x^{1/2}\) are eigen vectors and eigen values of the Laplacian matrix of X. This is the output of getNetMatrix.
- Sy: \(Q_y^{-1/2}E_y\Gamma_y^{1/2}\) where \(E_y\) and \(\Gamma_y^{1/2}\) are eigen vectors and eigen values of the Laplacian matrix of Y. This is the output of getNetMatrix.
- lambdas: 4-by-1 vector, the first and third values are sparsity penalty parameter for variables in X and Y each and the second and fourth values are network penalty parameters for variables in X and Y each.

Value

This function returns a list object.

- call: the matched call.
- res.opta: the estimated loading vector of transformed matrix of X.
- res.optb: the estimated loading vector of transformed matrix of Y.

See Also

see also oCIA, getNetMatrix, ssCIA.cv.

Examples

data("demoData"); attach(demoData)
X <- setX[1:n,]; nx <- sweep(X, 2, apply(X, 2, mean))
Y <- setY[1:n,]; ny <- sweep(Y, 2, apply(Y, 2, mean))
result.cia <- oCIA(nx, ny, D, Qx, Qy)
netmats <- getNetMatrix(p, q, eX, eY, Qx, Qy)
tlx <- netmats$Sx # tilde(l)_x
tly <- netmats$Sy # tilde(l)_y
result.sscia <- fitsscia(nx, ny, result.cia$result.opta[,]1, result.cia$result.optb[,]1, tlx, tly, c(6, 1, 6, 0.5))
**Description**

This function generates an object that is used as an input for the function `plot.pcia` that generates CIA plots.

**Usage**

```r
genpInput(respcia, namesX, namesY, samplename, class, X, Y, D, Qx, Qy)
```

**Arguments**

- `respcia`: an object of class `pcia`, result output of functions `fitscia`, `fitsscia`, `sciaNcv`, and `ssciaNcv`.
- `namesX`: variable name of data X.
- `namesY`: variable name of data Y.
- `samplename`: list of sample names.
- `class`: class information of samples.
- `X`: n-by-p data matrix, samples are rows and columns are features.
- `Y`: n-by-q data matrix, samples are rows and columns are features.
- `D`: n-by-n diagonal weight matrix for samples.
- `Qx`: p-by-p diagonal weight matrix for variables of X.
- `Qy`: q-by-q diagonal weight matrix for variables of Y.

**Value**

This function returns required coordinate informations and label informations used for generating CIA plots.

**See Also**

see also `plot.pcia`

**Examples**

```r
library(pcia)
data(list="NCI60", package="made4")
X <- t(NCI60$Ross)
Y <- t(NCI60$Affy)
D <- diag(dim(X)[1]); Qx <- diag(apply(abs(X), 2, sum) / sum(abs(X))); Qy <- diag(apply(abs(Y), 2, sum) / sum(abs(Y)));
nX <- sweep(X, 2, apply(X, 2, mean)); nY <- sweep(Y, 2, apply(Y, 2, mean))
result.cia <- oCIA(nX, nY, D, Qx, Qy)
pinput <- genpInput(result.cia, NCI60$Annot[,2], NCI60$Annot[,4], NCI60$classes[,1], NCI60$classes[,2], nX, nY)
```
**getNetMatrix**

*Generating Laplacian Matrix and its Eigendecomposition Results*

**Description**

This function calculates the Laplacian matrix and computes its eigendecomposition to generate an input matrix used in the ssCIA.

**Usage**

```r
getNetMatrix(p, q, eX, eY, Qx, Qy)
```

**Arguments**

- `p`: number of variables in the data `X`.
- `q`: number of variables in the data `Y`.
- `eX`: matrix with two columns. Each row contains the location of variables that are connected to each other in the data `X`.
- `eY`: matrix with two columns. Each row contains the location of variables that are connected to each other in the data `Y`.
- `Qx`: `p`-by-`p` diagonal weight matrix for variables of `X`.
- `Qy`: `q`-by-`q` diagonal weight matrix for variables of `Y`.

**Value**

This function returns a list object.

- `Sx`: $Q_x^{-1/2} E_x \Gamma_x^{1/2}$ where $E_x$ and $\Gamma_x^{1/2}$ are eigenvectors and eigenvalues of the Laplacian matrix of `X`. This is the output of `getNetMatrix`.
- `Sy`: $Q_y^{-1/2} E_y \Gamma_y^{1/2}$ where $E_y$ and $\Gamma_y^{1/2}$ are eigenvectors and eigenvalues of the Laplacian matrix of `Y`. This is the output of `getNetMatrix`.

**Examples**

```r
data(list="NCI60", package="made4")
X <- t(NCI60$Ross); Y <- t(NCI60$Affy)
Qx <- diag(apply(abs(X), 2, sum) / sum(abs(X)))
Qy <- diag(apply(abs(Y), 2, sum) / sum(abs(Y)))
nX <- sweep(X, 2, apply(X, 2, mean))
nY <- sweep(Y, 2, apply(Y, 2, mean))
data(NCI60$path)
netmats <- getNetMatrix(dim(nX)[2], dim(nY)[2], NCI60$path, NCI60$path, Qx, Qy)
```
NCI60path

Pathway information data of NCI60 data for Penalized Co-Inertia Analysis (CIA)

Description

Extracted edge information for NCI60 data contained in the package made4 from KEGG pathway database. Two columns stand for the vertices of all edges of existing graph in the data NCI60 dataset.

Usage

data(NCI60path)

References


Examples

data(NCI60path)

oCIA

Co-Inercia Analysis (CIA)

Description

This function conduct CIA.

Usage

oCIA(X, Y, D, Qx, Qy)

Arguments

X n-by-p data matrix, samples are rows and columns are features.
Y n-by-q data matrix, samples are rows and columns are features.
D n-by-n diagonal weight matrix for samples.
Qx p-by-p diagonal weight matrix for variables of X.
Qy q-by-q diagonal weight matrix for variables of Y.
Value

This function returns a list object.

- **call**: the matched call.
- **res.opta**: estimated loading vectors of of the transformed matrix $D^{1/2}XQ_x^{1/2}$.
- **res.optb**: estimated loading vectors of of the transformed matrix $D^{1/2}YQ_y^{1/2}$.
- **res.optu**: estimated loading vectors of of the matrix X.
- **res.optv**: estimated loading vectors of of the matrix Y.
- **ecoi**: estimated co-inertia values

Examples

```r
data("demoData"); attach(demodata)
X <- setX[1:n,]; nX <- sweep(X, 2, apply(X, 2, mean))
Y <- setY[1:n,]; nY <- sweep(Y, 2, apply(Y, 2, mean))
result.cia <- oCIA(nX, nY, D, Qx, Qy)
```

Description

This function generate output plots of CIA. Functions are based on the functions included in the R package ade4 and made4.

Usage

```r
## S3 method for class 'pcia'
plot(x, nlab = 10, labels = NULL, ...)
```

Arguments

- **nlab**: number of labels that is shown in the figure.
- **x**: inputs generated by the function genpinput contains estimated loaading vectors, normalized loading vectors, labels of variables of two datasets, group labels for samples.

Value

This function returns three different figures.

Examples

```r
library(pcia)
data(list="NCI60", package="made4")
X <- t(NCI60$Ross)
Y <- t(NCI60$Affy)
D <- diag(dim(X)[1]); Qx <- diag(apply(abs(X), 2, sum) / sum(abs(X))); Qy <- diag(apply(abs(Y), 2, sum) / sum(abs(Y)));
nX <- sweep(X, 2, apply(X, 2, mean)); nY <- sweep(Y, 2, apply(Y, 2, mean))
result.cia <- oCIA(nX, nY, D, Qx, Qy)
pinput <- genpinput(result.cia, NCI60$Annot[,2], NCI60$Annot[,4], NCI60$classes[,1], NCI60$classes[,2], nX, nY)
plot(pinput, nlab=5, labels = NULL)
```
sCIA.cv

Sparse Co-Inercia Analysis with Cross Validation for Choosing Optimal Tuning Parameters

Description

This function can be used to automatically select tuning parameters for sparse CIA.

Usage

sCIA.cv(x, Y, D, Qx, Qy, K = 1, TRUEval = NULL, nfold = 5, ngrid = 10, lambdarange = NULL, flag = TRUE)

Arguments

X n-by-p data matrix, samples are rows and columns are features.
Y n-by-q data matrix, samples are rows and columns are features.
D n-by-n diagonal weight matrix for samples.
Qx p-by-p diagonal weight matrix for variables of X.
Qy q-by-q diagonal weight matrix for variables of Y.
K number of loading pairs to be estimated.
TRUEval a list containing true loading vectors of X and Y in the simulation.
nfold number of subgroups that data is separated into for cross validation procedure.
ngrid number of possible tuning parameter values that will be searched.
lambdarange a vector contains four numbers. First and the second values are the minimum and maximum value of ranges for tuning parameters for X, the third and fourth are the minimum and maximum value of ranges for tuning parameters for Y.

Value

This function returns a list object. If this is used for a simulation and TRUEval is used, the output list contains CVobjs, OptTaus, and resa, resb, res.mea, res.num. If TRUEval is set as default, NULL, then the output list contains CVobjs, OptTaus, and resa, and resb.
call the matched call.
CVobjs cross validation objective values for all combination of tuning parameters.
OptTaus the chosen optimal tuning parameter pairs.
res.opta estimated loading vectors of of the transformed matrix $D^{1/2}XQ_x^{1/2}$.
res.optb estimated loading vectors of of the transformed matrix $D^{1/2}YQ_y^{1/2}$.
res.mea feature selection performance measures, sensitivity, specificity, MCC, and angle.
res.num count values, true positives, true negatives, false positives, and false negatives.

See Also

see also fitscia.
Examples

data("demoData"); attach(demoData)
X <- setX[1:n,]; nX <- sweep(X, 2, apply(X, 2, mean))
Y <- setY[1:n,]; nY <- sweep(Y, 2, apply(Y, 2, mean))
result.scia.cv <- sCIA.cv(nX, nY, D, Qx, Qy, K=2, nfold=5, ngrid=10, flag=FALSE)

ssCIA.cv

Structured Sparse Co-Inernia Analysis with Cross Validation for Chos-
ing Optimal Tuning Parameters

Description

This function can be used to automatically select tuning parameters for structured sparse CIA.

Usage

ssCIA.cv(X, Y, D, Qx, Qy, Sx, Sy, K = 1, nfold = 5, TRUEval = NULL,
ngrid = c(10, 3, 10, 3), lambdarange = NULL, lambda2range = NULL,
flag = TRUE)

Arguments

X n-by-p data matrix, samples are rows and columns are features.
Y n-by-q data matrix, samples are rows and columns are features.
D n-by-n diagonal weight matrix for samples.
Qx p-by-p diagonal weight matrix for variables of X.
Qy q-by-q diagonal weight matrix for variables of Y.
Sx \( Q_x^{-1/2} E_x \Gamma_x^{1/2} \) where \( E_x \) and \( \Gamma_x^{1/2} \) are eigen vectors and eigen values of the Laplacian matrix of X. This is the output of getNetMatrix.
Sy \( Q_y^{-1/2} E_y \Gamma_y^{1/2} \) where \( E_y \) and \( \Gamma_y^{1/2} \) are eigen vectors and eigen values of the Laplacian matrix of Y. This is the output of getNetMatrix.
K number of loading pairs to be estimated.
nfold number of subgroups that data is separated into for cross validation procedure.
TRUEval a list containing true loading vectors of X and Y in the simulation.
ngrid number of possible tuning parameter values that will be searched.
lambdarange a vector contains four numbers he range of sparsity penalty parameters. genGridPT function calculates grid points based on this range. Default values are NULL, in this case genGridPT function calculate range of tuning parameters from the result of lasso regression of glm package. Users can set the range of tuning parameters by putting desirable values. First and the second values are the minimum and maximum value of ranges for tuning parameters for X, the third and fourth area the minimum and maximum value of ranges for tuning parameters for Y.
lambda2range a vector contains four numbers the range of network penalty parameters. genGridPT function calculates grid points based on this range. Default values are NULL, in this case genGridPT function use three values (0.1, 0.5, 1) as a grid point. Users can set the range of tuning parameters by putting desirable values. First and the second values are the minimum and maximum value of ranges for tuning parameters for X, the third and fourth area the minimum and maximum value of ranges for tuning parameters for Y.
Value

This function returns a list object. If this is used for a simulation and TRUEval is used, the output list contains CVobj, OptTaus, and resa, resb, res.me, res.num. If TRUEval is set as default, NULL, then the output list contains CVobj, OptTaus, and resa, and resb.

call the matched call.
CVobj cross validation objective values for all combination of tuning parameters.
OptTaus the chosen optimal tuning parameter pairs.
resa estimated loading vectors of of the transformated matrix $D^{1/2}XQx^{1/2}$.
resb estimated loading vectors of of the transformated matrix $D^{1/2}YQy^{1/2}$.
res.me feature selection performance measures, sensitivity, specificity, MCC, and angle.
res.num count values, true positives, true negatives, false positives, and false negatives.

See Also

see also ocia, getNetMatrix, fitsscia.

Examples

data("demoData"); attach(demoData)
X <- setX[1:n.j]; nX <- sweep(X, 2, apply(X, 2, mean))
Y <- setY[1:n.j]; nY <- sweep(Y, 2, apply(Y, 2, mean))
netmats <- getNetMatrix(p, q, eX, eY, Qx, Qy)
tlx <- netmats$tx # tilde{l}_x
tly <- netmats$sy # tilde{l}_y
result.ssCIA.cv <- ssCIA.cv(nX, nY, D, Qx, Qy, tlx, tly, K=2, nfold=5, ngrid=c(5, 5, 5, 5), flag=FALSE)
Index

demoData, 2
fitscia, 3, 5, 9
fitsscia, 4, 5, 11

genInput, 5
getNetMatrix, 4, 6, 11

NCI60path, 7
ocIA, 3, 4, 7, 11
plot.pcia, 5, 8

sCIA.cv, 3, 5, 9
ssCIA.cv, 4, 5, 10