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 arguement to enchance 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` *Demo Dataset for Penalized Co-Inertia Analysis (CIA)*

Description

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

Usage

```
data(demoData)
```

Details

- p = 400. dimension of feateres in the data `X`
- q = 500. dimension of features in the data `Y`
- n = 200. sample size for one simulated data.
- nmcsample = 10. number of Monte Carlo simulated data.
- K=1. number of loading pairs to be estimated.
- setX. (n*nmcsample)-by-p data matrix
- setY. (n*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 -Omics data, Min et al. (2018)

Examples

```
data(demoData)
```

fitscia — *Sparse Co-Inercia Analysis*

Description

This function conduct 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 transformeddata 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 of transformated matrix of X.
res.optb	the estimated loading vector of of transformated matrix of Y.

See Also

see also oCIA, sCIA.cv.

Examples

```
library(pCIA)
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)
result.scia <- fitscia(nX, nY, result.cia$res.opta[,1], result.cia$res.optb[,1], c(6, 6))
```

fitsscia — *Structured Sparse Co-Inercia Analysis*

Description

This function conduct 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 transformeddata 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`	$Qx^{-1/2}E_x\Gamma_x^{1/2}$ 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`	$Qy^{-1/2}E_y\Gamma_y^{1/2}$ 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 of transformated matrix of `X`.
`res.optb`	the estimated loading vector of of transformated 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$res.opta[,1], result.cia$res.optb[,1], tLx, tLy, c(6, 1, 6, 0.5))
```

genpInput *Generate Co-inertia Analysis Figure*

Description

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

Usage

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

Arguments

respcia	an object of class pcia, result output of functions fitscia,fitsscia, sCIA.cv, and ssCIA.cv.
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

```
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(
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,
```

getNetMatrix *Generating Laplacian Matrix and its Eigendecompistion Results*

Description

This function calculate Laplacian matrix and computes the Eigendecompistion of the Laplacian matrix to generate input matrix used in the ssCIA.

Usage

```
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 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.
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	$Qx^{-1/2}E_x\Gamma_x^{1/2}$ 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	$Qy^{-1/2}E_y\Gamma_y^{1/2}$ 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.

Examples

```
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(NCI60path)
netmats <- getNetMatrix(dim(nX)[2], dim(nY)[2], NCI60path, NCI60path, Qx, Qy)
```

`NCI60path` *Patheway 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 columms stands for the vertices of all edges of existing graph in the data NCI60 dataset.

Usage

```
data(NCI60path)
```

References

Kanehisa, Furumichi, M., Tanabe, M., Sato, Y., and Morishima, K.; KEGG: new perspectives on genomes, pathways, diseases and drugs. Nucleic Acids Res. 45, D353-D361 (2017)

AC C, J T, G P and DG H (2005). "MADE4:an R package for multivariate analysis of gene expression data." Bioinformatics, 21(11), pp. 2789-90.

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 transformated matrix $D^{1/2}XQx^{1/2}$.
res.optb	estimated loading vectors of of the transformated matrix $D^{1/2}YQy^{1/2}$.
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

```
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)
```

---

plot.pcia — *Generate Co-inertia Analysis Figure*

Description

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

Usage

```
## 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

```
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(
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,
plot(pinput, nlab=5, labels = NULL)
```

`sCIA.cv` *Sparse Co-Inercia Analysis with Cross Validation for Chosing 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 paramter 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 tuniing parameters for `X`, the third and fourth are the minimum and maximum value of ranges for tuniing 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 transformated matrix $D^{1/2}XQx^{1/2}$.
`res.optb`	estimated loading vectors of of the transformated matrix $D^{1/2}YQy^{1/2}$.
`res.mea`	feature selction 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-Inercia Analysis with Cross Validation for Chosing 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`	$Qx^{-1/2}E_x\Gamma_x^{1/2}$ 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`	$Qy^{-1/2}E_y\Gamma_y^{1/2}$ 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 paramter 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 ranage. 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 ranage. 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 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.
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.mea	feature selction 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,]; nX <- sweep(X, 2, apply(X, 2, mean))
Y <- setY[1:n,]; nY <- sweep(Y, 2, apply(Y, 2, mean))
netmats <- getNetMatrix(p, q, eX, eY, Qx, Qy)
tLx <- netmats$Sx # tilde{L}_x
tLy <- netmats$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