Package 'fabia'

November 12, 2026

Title FABIA: Factor Analysis for Bicluster Acquisition

Version 2.58.0

Date 2020-01-30

Author Sepp Hochreiter <hochreit@bioinf.jku.at>

Maintainer Andreas Mitterecker <mitterecker@bioinf.jku.at>

Depends R (>= 3.6.0), Biobase

Imports methods, graphics, grDevices, stats, utils

Description Biclustering by ``Factor Analysis for Bicluster Acquisition'' (FABIA). FABIA is a model-based technique for biclustering, that is clustering rows and columns simultaneously. Biclusters are found by factor analysis where both the factors and the loading matrix are sparse. FABIA is a multiplicative model that extracts linear dependencies between samples and feature patterns. It captures realistic non-Gaussian data distributions with heavy tails as observed in gene expression measurements. FABIA utilizes well understood model selection techniques like the EM algorithm and variational approaches and is embedded into a Bayesian framework. FABIA ranks biclusters according to their information content and separates spurious biclusters from true biclusters. The code is written in C.

License LGPL (>= 2.1)

Collate AllClasses.R AllGenerics.R fabia.R methods-Factorization-class.R zzz.R

URL http://www.bioinf.jku.at/software/fabia/fabia.html

biocViews StatisticalMethod, Microarray, DifferentialExpression, MultipleComparison, Clustering, Visualization

git_url https://git.bioconductor.org/packages/fabia

git_branch RELEASE_3_23

git_last_commit f9e447c

git_last_commit_date 2026-04-28

Repository Bioconductor 3.23

Date/Publication 2026-11-11

Contents

estimateMode . . . 2
extractBic . . . 4
extractPlot . . . 7
fabi . . . 10
fabia . . . 13
fabiaDemo . . . 20
fabiap . . . 21
fabias . . . 27
fabiasp . . . 34
fabiaVersion . . . 38
Factorization-class . . . 39
makeFabiaData . . . 47
makeFabiaDataBlocks . . . 50
makeFabiaDataBlocksPos . . . 52
makeFabiaDataPos . . . 55
matrixImagePlot . . . 57
mfsc . . . 59
nmfdiv . . . 65
nmfeu . . . 67
nmfsc . . . 69
plotBicluster . . . 71
projFunc . . . 74
projFuncPos . . . 75
readSamplesSpfabia . . . 76
readSpfabiaResult . . . 78
samplesPerFeature . . . 79
spfabia . . . 80

Index **84**

`estimateMode` *Estimation of the modes of the rows of a matrix*

Description

`estimateMode`: R implementation of `estimateMode`.

Usage

```
estimateMode(X,maxiter=50,tol=0.001,alpha=0.1,a1=4.0,G1=FALSE)
```

Arguments

X	matrix of which the modes of the rows are estimated.
maxiter	maximal number of iterations; default = 50.
tol	tolerance for stopping; default = 0.001.
alpha	learning rate; default = 0.1.
a1	parameter of the width of the given distribution; default = 4.
G1	kind of distribution, TRUE: G1, FALSE: G2; default = FALSE.

Details

The mode is estimated by contrast functions G1

$$(1/a_1) * \ln(\cosh(a1 * x))$$

or G2

$$-(1/a_1) * \exp(-1/2 * x * x)$$

The estimation is performed by gradient descent initialized by the median.

Implementation in R.

Value

u	the vector of estimated modes.
xu	$X - u$ the mode centered data.

Author(s)

Sepp Hochreiter

References

A. Hyvaerinen, 'Fast and Robust Fixed-Point Algorithms for Independent Component Analysis', IEEE Transactions on Neural Networks 10(3):626-634, 1999.

See Also

fabia, fabias, fabiap, fabi, fabiasp, mfsc, nmfdiv, nmfeu, nmfsc, extractPlot, extractBic, plotBicluster, Factorization, projFuncPos, projFunc, estimateMode, makeFabiaData, makeFabiaDataBlocks, makeFabiaDataPos, makeFabiaDataBlocksPos, matrixImagePlot, fabiaDemo, fabiaVersion

Examples

```
#---------------
# DEMO
#---------------

dat <- makeFabiaDataBlocksPos(n = 100,l= 50,p = 10,f1 = 5,f2 = 5,
       of1 = 5,of2 = 10,sd_noise = 2.0,sd_z_noise = 0.2,mean_z = 2.0,
       sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)
```

```
X <- dat[[1]]

# modes <- estimateMode(X)

# modes$u - apply(X, 1, median)
```

`extractBic` *Extraction of Biclusters*

Description

`extractBic`: R implementation of `extractBic`.

Usage

```
extractBic(fact,thresZ=0.5,thresL=NULL)
```

Arguments

`fact`	object of the class `Factorization`.
`thresZ`	threshold for sample belonging to bicluster; default 0.5.
`thresL`	threshold for loading belonging to bicluster (if not given it is estimated).

Details

Essentially the model is the sum of outer products of vectors:

$$X = \sum_{i=1}^{p} \lambda_i z_i^T + U$$

where the number of summands p is the number of biclusters. The matrix factorization is

$$X = LZ + U$$

Here λ_i are from R^n, z_i from R^l, L from $R^{n \times p}$, Z from $R^{p \times l}$, and X, U from $R^{n \times l}$.
U is the Gaussian noise with a diagonal covariance matrix which entries are given by `Psi`.
The Z is locally approximated by a Gaussian with inverse variance given by `lapla`.
Using these values we can computer for each j the variance z_i given x_j. Here

$$x_j = Lz_j + u_j$$

This variance can be used to determine the information content of a bicluster.

The λ_i and z_i are used to extract the bicluster i, where a threshold determines which observations and which samples belong the the bicluster.

In `bic` the biclusters are extracted according to the largest absolute values of the component i, i.e. the largest values of λ_i and the largest values of z_i. The factors z_i are normalized to variance 1.

The components of `bic` are `binp`, `bixv`, `bixn`, `biypv`, and `biypn`.

`binp` give the size of the bicluster: number observations and number samples. `bixv` gives the values of the extracted observations that have absolute values above a threshold. They are sorted. `bixn` gives the extracted observation names (e.g. gene names). `biypv` gives the values of the extracted samples that have absolute values above a threshold. They are sorted. `biypn` gives the names of the extracted samples (e.g. sample names).

In `bicopp` the opposite clusters to the biclusters are given. Opposite means that the negative pattern is present.

The components of opposite clusters `bicopp` are `binn`, `bixv`, `bixn`, `biypnv`, and `biynn`.

`binp` give the size of the opposite bicluster: number observations and number samples. `bixv` gives the values of the extracted observations that have absolute values above a threshold. They are sorted. `bixn` gives the extracted observation names (e.g. gene names). `biynv` gives the values of the opposite extracted samples that have absolute values above a threshold. They are sorted. `biynn` gives the names of the opposite extracted samples (e.g. sample names).

That means the samples are divided into two groups where one group shows large positive values and the other group has negative values with large absolute values. That means a observation pattern can be switched on or switched off relative to the average value.

`numn` gives the indices of `bic` with components: `numng` = `bix` and `numnp` = `biypn`.

`numn` gives the indices of `bicopp` with components: `numng` = `bix` and `numnn` = `biynn`.

Implementation in R.

Value

`bic`	extracted biclusters.
`numn`	indexes for the extracted biclusters.
`bicopp`	extracted opposite biclusters.
`numnopp`	indexes for the extracted opposite biclusters.
`X`	scaled and centered data matrix.
`np`	number of biclusters.

Author(s)

Sepp Hochreiter

See Also

`fabia`, `fabias`, `fabiap`, `fabi`, `fabiasp`, `mfsc`, `nmfdiv`, `nmfeu`, `nmfsc`, `extractPlot`, `extractBic`, `plotBicluster`, `Factorization`, `projFuncPos`, `projFunc`, `estimateMode`, `makeFabiaData`, `makeFabiaDataBlocks`, `makeFabiaDataPos`, `makeFabiaDataBlocksPos`, `matrixImagePlot`, `fabiaDemo`, `fabiaVersion`

Examples

```
#---------------
# TEST
#---------------

dat <- makeFabiaDataBlocks(n = 100,l= 50,p = 3,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

X <- dat[[1]]
Y <- dat[[2]]


resEx <- fabia(X,3,0.01,20)

rEx <- extractBic(resEx)

rEx$bic[1,]
rEx$bic[2,]
rEx$bic[3,]


## Not run:
#---------------
# DEMO1
#---------------

dat <- makeFabiaDataBlocks(n = 1000,l= 100,p = 10,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

X <- dat[[1]]
Y <- dat[[2]]


resToy <- fabia(X,13,0.01,200)

rToy <- extractBic(resToy)

avini(resToy)

rToy$bic[1,]
rToy$bic[2,]
rToy$bic[3,]

#---------------
# DEMO2
#---------------


avail <- require(fabiaData)
```

```
if (!avail) {
    message("")
    message("")
    message("######################################################")
    message("Package 'fabiaData' is not available: please install.")
    message("######################################################")
} else {

data(Breast_A)

X <- as.matrix(XBreast)

resBreast <- fabia(X,5,0.1,200)

rBreast <- extractBic(resBreast)

avini(resBreast)

rBreast$bic[1,]
rBreast$bic[2,]
rBreast$bic[3,]
}


## End(Not run)
```

`extractPlot` *Plotting of Biclustering Results*

Description

`extractPlot`: R implementation of `extractPlot`.

Usage

```
extractPlot(fact,thresZ=0.5,ti="",thresL=NULL,Y=NULL,which=c(1,2,3,4,5,6))
```

Arguments

`fact`	object of the class `Factorization`.
`thresZ`	threshold for sample belonging to bicluster; default 0.5.
`thresL`	threshold for loading belonging to bicluster (estimated if not given).
`ti`	plot title; default "".
`Y`	noise free data matrix.
`which`	which plot is shown: 1=noise free data (if available), 2=data, 3=reconstructed data, 4=error, 5=absolute factors, 6=absolute loadings; default c(1,2,3,4,5,6).

Details

Essentially the model is the sum of outer products of vectors:

$$X = \sum_{i=1}^{p} \lambda_i z_i^T + U$$

where the number of summands p is the number of biclusters. The matrix factorization is

$$X = LZ + U$$

Here λ_i are from R^n, z_i from R^l, L from $R^{n \times p}$, Z from $R^{p \times l}$, and X, U from $R^{n \times l}$.

The hidden dimension p is used for kmeans clustering of λ_i and z_i.

The λ_i and z_i are used to extract the bicluster i, where a threshold determines which observations and which samples belong the the bicluster.

The method produces following plots depending what plots are chosen by the "which" variable:

“Y”: noise free data (if available), “X”: data, “LZ”: reconstructed data, “LZ-X”: error, “abs(Z)”: absolute factors, “abs(L)”: absolute loadings.

Implementation in R.

Value

Returns corresponding plots

Author(s)

Sepp Hochreiter

See Also

fabia, fabias, fabiap, fabi, fabiasp, spfabia, mfsc, nmfdiv, nmfeu, nmfsc, extractPlot, extractBic, plotBicluster, Factorization, projFuncPos, projFunc, estimateMode, makeFabiaData, makeFabiaDataBlocks, makeFabiaDataPos, makeFabiaDataBlocksPos, matrixImagePlot, fabiaDemo, fabiaVersion

Examples

```
#---------------
# TEST
#---------------

dat <- makeFabiaDataBlocks(n = 100,l= 50,p = 3,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

X <- dat[[1]]
Y <- dat[[2]]
```

```
resEx <- fabia(X,3,0.1,20)

extractPlot(resEx,ti="FABIA",Y=Y)



## Not run:
#---------------
# DEMO1
#---------------

dat <- makeFabiaDataBlocks(n = 1000,l= 100,p = 10,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

X <- dat[[1]]
Y <- dat[[2]]


resToy <- fabia(X,13,0.01,200)

extractPlot(resToy,ti="FABIA",Y=Y)

#---------------
# DEMO2
#---------------

avail <- require(fabiaData)

if (!avail) {
    message("")
    message("")
    message("########################################################")
    message("Package 'fabiaData' is not available: please install.")
    message("########################################################")
} else {

data(Breast_A)

X <- as.matrix(XBreast)

resBreast <- fabia(X,5,0.1,200)

extractPlot(resBreast,ti="FABIA Breast cancer(Veer)")

#sorting of predefined labels
CBreast
}


## End(Not run)
```

fabi *Factor Analysis for Bicluster Acquisition: Laplace Prior (FABI)*

Description

fabi: R implementation of fabia, therefore it is **slow**.

Usage

```
fabi(X,p=13,alpha=0.01,cyc=500,spl=0,spz=0.5,center=2,norm=1,lap=1.0)
```

Arguments

X	the data matrix.
p	number of hidden factors = number of biclusters; default = 13.
alpha	sparseness loadings (0-1.0); default = 0.01.
cyc	number of iterations; default = 500.
spl	sparseness prior loadings (0 - 2.0); default = 0 (Laplace).
spz	sparseness factors (0.5-2.0); default = 0.5 (Laplace).
center	data centering: 1 (mean), 2 (median), > 2 (mode), 0 (no); default = 2.
norm	data normalization: 1 (0.75-0.25 quantile), >1 (var=1), 0 (no); default = 1.
lap	minimal value of the variational parameter; default = 1.0.

Details

Biclusters are found by sparse factor analysis where *both* the factors and the loadings are sparse.

Essentially the model is the sum of outer products of vectors:

$$X = \sum_{i=1}^{p} \lambda_i z_i^T + U$$

where the number of summands p is the number of biclusters. The matrix factorization is

$$X = LZ + U$$

Here λ_i are from R^n, z_i from R^l, L from $R^{n \times p}$, Z from $R^{p \times l}$, and X, U from $R^{n \times l}$.

If the nonzero components of the sparse vectors are grouped together then the outer product results in a matrix with a nonzero block and zeros elsewhere.

We recommend to *normalize the components to variance one* in order to make the signal and noise comparable across components.

The model selection is performed by a variational approach according to Girolami 2001 and Palmer et al. 2006.

We included a prior on the parameters and minimize a lower bound on the posterior of the parameters given the data. The update of the loadings includes an additive term which pushes the loadings toward zero (Gaussian prior leads to an multiplicative factor).

The code is implemented in R, therefore it is **slow**.

Value

object of the class `Factorization`. Containing `LZ` (estimated noise free data LZ), `L` (loadings L), `Z` (factors Z), `U` (noise $X - LZ$), `center` (centering vector), `scaleData` (scaling vector), `X` (centered and scaled data X), `Psi` (noise variance σ), `lapla` (variational parameter), `avini` (the information which the factor z_{ij} contains about x_j averaged over j) `xavini` (the information which the factor z_j contains about x_j) `ini` (for each j the information which the factor z_{ij} contains about x_j).

Author(s)

Sepp Hochreiter

References

S. Hochreiter et al., 'FABIA: Factor Analysis for Bicluster Acquisition', Bioinformatics 26(12):1520-1527, 2010. http://bioinformatics.oxfordjournals.org/cgi/content/abstract/btq227

Mark Girolami, 'A Variational Method for Learning Sparse and Overcomplete Representations', Neural Computation 13(11): 2517-2532, 2001.

J. Palmer, D. Wipf, K. Kreutz-Delgado, B. Rao, 'Variational EM algorithms for non-Gaussian latent variable models', Advances in Neural Information Processing Systems 18, pp. 1059-1066, 2006.

See Also

`fabia`, `fabias`, `fabiap`, `spfabia`, `fabi`, `fabiasp`, `mfsc`, `nmfdiv`, `nmfeu`, `nmfsc`, `extractPlot`, `extractBic`, `plotBicluster`, `Factorization`, `projFuncPos`, `projFunc`, `estimateMode`, `makeFabiaData`, `makeFabiaDataBlocks`, `makeFabiaDataPos`, `makeFabiaDataBlocksPos`, `matrixImagePlot`, `fabiaDemo`, `fabiaVersion`

Examples

```
#---------------
# TEST
#---------------

dat <- makeFabiaDataBlocks(n = 100,l= 50,p = 3,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

X <- dat[[1]]
Y <- dat[[2]]

resEx <- fabi(X,3,0.01,20)


## Not run:
#---------------
# DEMO1
#---------------
```

```
dat <- makeFabiaDataBlocks(n = 1000,l= 100,p = 10,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

X <- dat[[1]]
Y <- dat[[2]]


resToy <- fabi(X,13,0.01,200)

extractPlot(resToy,ti="FABI",Y=Y)

#---------------
# DEMO2
#---------------

avail <- require(fabiaData)

if (!avail) {
    message("")
    message("")
    message("#####################################################")
    message("Package 'fabiaData' is not available: please install.")
    message("#####################################################")
} else {

data(Breast_A)

X <- as.matrix(XBreast)

resBreast <- fabi(X,5,0.1,200)

extractPlot(resBreast,ti="FABI Breast cancer(Veer)")

#sorting of predefined labels
CBreast
}

#---------------
# DEMO3
#---------------


avail <- require(fabiaData)

if (!avail) {
    message("")
    message("")
    message("#####################################################")
    message("Package 'fabiaData' is not available: please install.")
    message("#####################################################")
} else {
```

```
data(Multi_A)

X <- as.matrix(XMulti)

resMulti <- fabi(X,5,0.1,200)

extractPlot(resMulti,ti="FABI Multiple tissues(Su)")

#sorting of predefined labels
CMulti
}


#---------------
# DEMO4
#---------------


avail <- require(fabiaData)

if (!avail) {
    message("")
    message("")
    message("######################################################")
    message("Package 'fabiaData' is not available: please install.")
    message("######################################################")
} else {


data(DLBCL_B)

X <- as.matrix(XDLBCL)


resDLBCL <- fabi(X,5,0.1,200)

extractPlot(resDLBCL,ti="FABI Lymphoma(Rosenwald)")

#sorting of predefined labels
CDLBCL
}


## End(Not run)
```

`fabia` *Factor Analysis for Bicluster Acquisition: Laplace Prior (FABIA)*

Description

`fabia`: C implementation of `fabia`.

Usage

```
fabia(X,p=13,alpha=0.01,cyc=500,spl=0,spz=0.5,non_negative=0,random=1.0,center=2,norm=1,scale=0.0,l
```

Arguments

X	the data matrix.
p	number of hidden factors = number of biclusters; default = 13.
alpha	sparseness loadings (0 - 1.0); default = 0.01.
cyc	number of iterations; default = 500.
spl	sparseness prior loadings (0 - 2.0); default = 0 (Laplace).
spz	sparseness factors (0.5 - 2.0); default = 0.5 (Laplace).
non_negative	Non-negative factors and loadings if non_negative > 0; default = 0.
random	<=0: by SVD, >0: random initialization of loadings in [-random,random]; default = 1.0.
center	data centering: 1 (mean), 2 (median), > 2 (mode), 0 (no); default = 2.
norm	data normalization: 1 (0.75-0.25 quantile), >1 (var=1), 0 (no); default = 1.
scale	loading vectors are scaled in each iteration to the given variance. 0.0 indicates non scaling; default = 0.0.
lap	minimal value of the variational parameter; default = 1.0
nL	maximal number of biclusters at which a row element can participate; default = 0 (no limit)
lL	maximal number of row elements per bicluster; default = 0 (no limit)
bL	cycle at which the nL or lL maximum starts; default = 0 (start at the beginning)

Details

Biclusters are found by sparse factor analysis where *both* the factors and the loadings are sparse.

Essentially the model is the sum of outer products of vectors:

$$X = \sum_{i=1}^{p} \lambda_i z_i^T + U$$

where the number of summands p is the number of biclusters. The matrix factorization is

$$X = LZ + U$$

Here λ_i are from R^n, z_i from R^l, L from $R^{n \times p}$, Z from $R^{p \times l}$, and X, U from $R^{n \times l}$.

If the nonzero components of the sparse vectors are grouped together then the outer product results in a matrix with a nonzero block and zeros elsewhere.

The model selection is performed by a variational approach according to Girolami 2001 and Palmer et al. 2006.

We included a prior on the parameters and minimize a lower bound on the posterior of the parameters given the data. The update of the loadings includes an additive term which pushes the loadings toward zero (Gaussian prior leads to an multiplicative factor).

The code is implemented in C.

Value

object of the class `Factorization`. Containing `LZ` (estimated noise free data LZ), `L` (loadings L), `Z` (factors Z), `U` (noise: $X - LZ$), `center` (centering vector), `scaleData` (scaling vector), `X` (centered and scaled data X), `Psi` (noise variance σ), `lapla` (variational parameter), `avini` (the information which the factor z_{ij} contains about x_j averaged over j) `xavini` (the information which the factor z_j contains about x_j) `ini` (for each j the information which the factor z_{ij} contains about x_j).

Author(s)

Sepp Hochreiter

References

S. Hochreiter et al., 'FABIA: Factor Analysis for Bicluster Acquisition', Bioinformatics 26(12):1520-1527, 2010. http://bioinformatics.oxfordjournals.org/cgi/content/abstract/btq227

Mark Girolami, 'A Variational Method for Learning Sparse and Overcomplete Representations', Neural Computation 13(11): 2517-2532, 2001.

J. Palmer, D. Wipf, K. Kreutz-Delgado, B. Rao, 'Variational EM algorithms for non-Gaussian latent variable models', Advances in Neural Information Processing Systems 18, pp. 1059-1066, 2006.

See Also

`fabia`, `fabias`, `fabiap`, `spfabia`, `readSpfabiaResult`, `fabi`, `fabiasp`, `mfsc`, `nmfdiv`, `nmfeu`, `nmfsc`, `extractPlot`, `extractBic`, `plotBicluster`, `Factorization`, `projFuncPos`, `projFunc`, `estimateMode`, `makeFabiaData`, `makeFabiaDataBlocks`, `makeFabiaDataPos`, `makeFabiaDataBlocksPos`, `matrixImagePlot`, `fabiaDemo`, `fabiaVersion`

Examples

```
#---------------
# TEST
#---------------

dat <- makeFabiaDataBlocks(n = 100,l= 50,p = 3,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

X <- dat[[1]]
Y <- dat[[2]]


resEx <- fabia(X,3,0.01,50)


## Not run:
#-----------------
# DEMO1: Toy Data
#-----------------
```

```
n = 1000
l= 100
p = 10

dat <- makeFabiaDataBlocks(n = n,l= l,p = p,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

X <- dat[[1]]
Y <- dat[[2]]
ZC <- dat[[3]]
LC <- dat[[4]]

gclab <- rep.int(0,l)
gllab <- rep.int(0,n)
clab <- as.character(1:l)
llab <- as.character(1:n)
for (i in 1:p){
 for (j in ZC[i]){
     clab[j] <- paste(as.character(i),"_",clab[j],sep="")
 }
 for (j in LC[i]){
     llab[j] <- paste(as.character(i),"_",llab[j],sep="")
 }
 gclab[unlist(ZC[i])] <- gclab[unlist(ZC[i])] + p^i
 gllab[unlist(LC[i])] <- gllab[unlist(LC[i])] + p^i
}


groups <- gclab



#### FABIA

resToy1 <- fabia(X,13,0.01,400)

extractPlot(resToy1,ti="FABIA",Y=Y)

raToy1 <- extractBic(resToy1)

if ((raToy1$bic[[1]][1]>1) && (raToy1$bic[[1]][2])>1) {
    plotBicluster(raToy1,1)
}
if ((raToy1$bic[[2]][1]>1) && (raToy1$bic[[2]][2])>1) {
plotBicluster(raToy1,2)
}
if ((raToy1$bic[[3]][1]>1) && (raToy1$bic[[3]][2])>1) {
plotBicluster(raToy1,3)
}
if ((raToy1$bic[[4]][1]>1) && (raToy1$bic[[4]][2])>1) {
plotBicluster(raToy1,4)
```

```
}



colnames(X(resToy1)) <- clab

rownames(X(resToy1)) <- llab


plot(resToy1,dim=c(1,2),label.tol=0.1,col.group = groups,lab.size=0.6)
plot(resToy1,dim=c(1,3),label.tol=0.1,col.group = groups,lab.size=0.6)
plot(resToy1,dim=c(2,3),label.tol=0.1,col.group = groups,lab.size=0.6)



#----------------------------------------
# DEMO2: Laura van't Veer's gene expression
#        data set for breast cancer
#----------------------------------------

avail <- require(fabiaData)

if (!avail) {
    message("")
    message("")
    message("#####################################################")
    message("Package 'fabiaData' is not available: please install.")
    message("#####################################################")
} else {

data(Breast_A)

X <- as.matrix(XBreast)


resBreast1 <- fabia(X,5,0.1,400)

extractPlot(resBreast1,ti="FABIA Breast cancer(Veer)")


raBreast1 <- extractBic(resBreast1)

if ((raBreast1$bic[[1]][1]>1) && (raBreast1$bic[[1]][2])>1) {
    plotBicluster(raBreast1,1)
}
if ((raBreast1$bic[[2]][1]>1) && (raBreast1$bic[[2]][2])>1) {
    plotBicluster(raBreast1,2)
}
if ((raBreast1$bic[[3]][1]>1) && (raBreast1$bic[[3]][2])>1) {
    plotBicluster(raBreast1,3)
}
if ((raBreast1$bic[[4]][1]>1) && (raBreast1$bic[[4]][2])>1) {
    plotBicluster(raBreast1,4)
```

```
}


plot(resBreast1,dim=c(1,2),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast1,dim=c(1,3),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast1,dim=c(1,4),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast1,dim=c(1,5),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast1,dim=c(2,3),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast1,dim=c(2,4),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast1,dim=c(2,5),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast1,dim=c(3,4),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast1,dim=c(3,5),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast1,dim=c(4,5),label.tol=0.03,col.group=CBreast,lab.size=0.6)

}


#-----------------------------------
# DEMO3: Su's multiple tissue types
#        gene expression data set
#-----------------------------------


avail <- require(fabiaData)

if (!avail) {
    message("")
    message("")
    message("#####################################################")
    message("Package 'fabiaData' is not available: please install.")
    message("#####################################################")
} else {


data(Multi_A)

X <- as.matrix(XMulti)

resMulti1 <- fabia(X,5,0.06,300,norm=2)

extractPlot(resMulti1,ti="FABIA Multiple tissues(Su)")

raMulti1 <- extractBic(resMulti1)

if ((raMulti1$bic[[1]][1]>1) && (raMulti1$bic[[1]][2])>1) {
    plotBicluster(raMulti1,1)
}
if ((raMulti1$bic[[2]][1]>1) && (raMulti1$bic[[2]][2])>1) {
    plotBicluster(raMulti1,2)
}
if ((raMulti1$bic[[3]][1]>1) && (raMulti1$bic[[3]][2])>1) {
    plotBicluster(raMulti1,3)
}
```

```
if ((raMulti1$bic[[4]][1]>1) && (raMulti1$bic[[4]][2])>1) {
    plotBicluster(raMulti1,4)
}

plot(resMulti1,dim=c(1,2),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti1,dim=c(1,3),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti1,dim=c(1,4),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti1,dim=c(1,5),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti1,dim=c(2,3),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti1,dim=c(2,4),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti1,dim=c(2,5),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti1,dim=c(3,4),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti1,dim=c(3,5),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti1,dim=c(4,5),label.tol=0.01,col.group=CMulti,lab.size=0.6)

}


#----------------------------------------
# DEMO4: Rosenwald's diffuse large-B-cell
#        lymphoma gene expression data set
#----------------------------------------


avail <- require(fabiaData)

if (!avail) {
    message("")
    message("")
    message("#####################################################")
    message("Package 'fabiaData' is not available: please install.")
    message("#####################################################")
} else {


data(DLBCL_B)

X <- as.matrix(XDLBCL)

resDLBCL1 <- fabia(X,5,0.1,400,norm=2)

extractPlot(resDLBCL1,ti="FABIA Lymphoma(Rosenwald)")

raDLBCL1 <- extractBic(resDLBCL1)

if ((raDLBCL1$bic[[1]][1]>1) && (raDLBCL1$bic[[1]][2])>1) {
    plotBicluster(raDLBCL1,1)
}
if ((raDLBCL1$bic[[2]][1]>1) && (raDLBCL1$bic[[2]][2])>1) {
    plotBicluster(raDLBCL1,2)
}
if ((raDLBCL1$bic[[3]][1]>1) && (raDLBCL1$bic[[3]][2])>1) {
    plotBicluster(raDLBCL1,3)
```

```
}
if ((raDLBCL1$bic[[4]][1]>1) && (raDLBCL1$bic[[4]][2])>1) {
    plotBicluster(raDLBCL1,4)
}

plot(resDLBCL1,dim=c(1,2),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL1,dim=c(1,3),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL1,dim=c(1,4),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL1,dim=c(1,5),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL1,dim=c(2,3),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL1,dim=c(2,4),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL1,dim=c(2,5),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL1,dim=c(3,4),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL1,dim=c(3,5),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL1,dim=c(4,5),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)

}


## End(Not run)
```

`fabiaDemo` *Demos for fabia*

Description

`fabiaDemo` calls the demo codes for fabia.

Usage

```
fabiaDemo()
```

Value

Calls the demo codes for fabia

Author(s)

Sepp Hochreiter

See Also

`fabia`, `fabias`, `fabiap`, `fabi`, `fabiasp`, `mfsc`, `nmfdiv`, `nmfeu`, `nmfsc`, `extractPlot`, `extractBic`, `plotBicluster`, `Factorization`, `projFuncPos`, `projFunc`, `estimateMode`, `makeFabiaData`, `makeFabiaDataBlocks`, `makeFabiaDataPos`, `makeFabiaDataBlocksPos`, `matrixImagePlot`, `fabiaDemo`, `fabiaVersion`

Examples

```
## Not run:
# interactive
fabiaDemo()

## End(Not run)
```

---

fabiap *Factor Analysis for Bicluster Acquisition: Post-Projection (FABIAP)*

Description

fabiap: C implementation of fabiap.

Usage

```
fabiap(X,p=13,alpha=0.01,cyc=500,spl=0,spz=0.5,sL=0.6,sZ=0.6,non_negative=0,random=1.0,center=2,nor
```

Arguments

X	the data matrix.
p	number of hidden factors = number of biclusters; default = 13.
alpha	sparseness loadings (0-1.0); default = 0.01.
cyc	number of iterations; default = 500.
spl	sparseness prior loadings (0 - 2.0); default = 0 (Laplace).
spz	sparseness factors (0.5 - 2.0); default = 0.5 (Laplace).
sL	final sparseness loadings; default = 0.6.
sZ	final sparseness factors; default = 0.6.
non_negative	Non-negative factors and loadings if non_negative > 0; default = 0.
random	<=0: by SVD, >0: random initialization of loadings in [-random,random]; default = 1.0.
center	data centering: 1 (mean), 2 (median), > 2 (mode), 0 (no); default = 2.
norm	data normalization: 1 (0.75-0.25 quantile), >1 (var=1), 0 (no); default = 1.
scale	loading vectors are scaled in each iteration to the given variance. 0.0 indicates non scaling; default = 0.0.
lap	minimal value of the variational parameter; default = 1.0.
nL	maximal number of biclusters at which a row element can participate; default = 0 (no limit)
lL	maximal number of row elements per bicluster; default = 0 (no limit)
bL	cycle at which the nL or lL maximum starts; default = 0 (start at the beginning)

Details

Biclusters are found by sparse factor analysis where *both* the factors and the loadings are sparse. Post-processing by projecting the final results to a given sparseness criterion.

Essentially the model is the sum of outer products of vectors:

$$X = \sum_{i=1}^{p} \lambda_i z_i^T + U$$

where the number of summands p is the number of biclusters. The matrix factorization is

$$X = LZ + U$$

Here λ_i are from R^n, z_i from R^l, L from $R^{n \times p}$, Z from $R^{p \times l}$, and X, U from $R^{n \times l}$.

If the nonzero components of the sparse vectors are grouped together then the outer product results in a matrix with a nonzero block and zeros elsewhere.

The model selection is performed by a variational approach according to Girolami 2001 and Palmer et al. 2006.

We included a prior on the parameters and minimize a lower bound on the posterior of the parameters given the data. The update of the loadings includes an additive term which pushes the loadings toward zero (Gaussian prior leads to an multiplicative factor).

Post-processing: The final results of the loadings and the factors are projected to a sparse vector according to Hoyer, 2004: given an l_1-norm and an l_2-norm minimize the Euclidean distance to the original vector (currently the l_2-norm is fixed to 1). The projection is a convex quadratic problem which is solved iteratively where at each iteration at least one component is set to zero. Instead of the l_1-norm a sparseness measurement is used which relates the l_1-norm to the l_2-norm:

The code is implemented in C and the projection in R.

Value

object of the class `Factorization`. Containing `LZ` (estimated noise free data LZ), `L` (loadings L), `Z` (factors Z), `U` (noise $X - LZ$), `center` (centering vector), `scaleData` (scaling vector), `X` (centered and scaled data X), `Psi` (noise variance σ), `lapla` (variational parameter), `avini` (the information which the factor z_{ij} contains about x_j averaged over j) `xavini` (the information which the factor z_j contains about x_j) `ini` (for each j the information which the factor z_{ij} contains about x_j).

Author(s)

Sepp Hochreiter

References

S. Hochreiter et al., 'FABIA: Factor Analysis for Bicluster Acquisition', Bioinformatics 26(12):1520-1527, 2010. http://bioinformatics.oxfordjournals.org/cgi/content/abstract/btq227

Mark Girolami, 'A Variational Method for Learning Sparse and Overcomplete Representations', Neural Computation 13(11): 2517-2532, 2001.

J. Palmer, D. Wipf, K. Kreutz-Delgado, B. Rao, 'Variational EM algorithms for non-Gaussian latent variable models', Advances in Neural Information Processing Systems 18, pp. 1059-1066, 2006.

Patrik O. Hoyer, 'Non-negative Matrix Factorization with Sparseness Constraints', Journal of Machine Learning Research 5:1457-1469, 2004.

See Also

fabia, fabias, fabiap, spfabia, fabi, fabiasp, mfsc, nmfdiv, nmfeu, nmfsc, extractPlot, extractBic, plotBicluster, Factorization, projFuncPos, projFunc, estimateMode, makeFabiaData, makeFabiaDataBlocks, makeFabiaDataPos, makeFabiaDataBlocksPos, matrixImagePlot, fabiaDemo, fabiaVersion

Examples

```
#---------------
# TEST
#---------------

dat <- makeFabiaDataBlocks(n = 100,l= 50,p = 3,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

X <- dat[[1]]
Y <- dat[[2]]

resEx <- fabiap(X,3,0.1,50)


## Not run:

#-----------------
# DEMO1: Toy Data
#-----------------

n = 1000
l= 100
p = 10

dat <- makeFabiaDataBlocks(n = n,l= l,p = p,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

X <- dat[[1]]
Y <- dat[[2]]
ZC <- dat[[3]]
LC <- dat[[4]]

gclab <- rep.int(0,l)
gllab <- rep.int(0,n)
clab <- as.character(1:l)
llab <- as.character(1:n)
for (i in 1:p){
```

```
 for (j in ZC[i]){
     clab[j] <- paste(as.character(i),"_",clab[j],sep="")
 }
 for (j in LC[i]){
     llab[j] <- paste(as.character(i),"_",llab[j],sep="")
 }
 gclab[unlist(ZC[i])] <- gclab[unlist(ZC[i])] + p^i
 gllab[unlist(LC[i])] <- gllab[unlist(LC[i])] + p^i
}


groups <- gclab

#### FABIAP

resToy3 <- fabiap(X,13,0.1,400)

extractPlot(resToy3,ti="FABIAP",Y=Y)

raToy3 <- extractBic(resToy3)

if ((raToy3$bic[[1]][1]>1) && (raToy3$bic[[1]][2])>1) {
    plotBicluster(raToy3,1)
}
if ((raToy3$bic[[2]][1]>1) && (raToy3$bic[[2]][2])>1) {
    plotBicluster(raToy3,2)
}
if ((raToy3$bic[[3]][1]>1) && (raToy3$bic[[3]][2])>1) {
    plotBicluster(raToy3,3)
}
if ((raToy3$bic[[4]][1]>1) && (raToy3$bic[[4]][2])>1) {
    plotBicluster(raToy3,4)
}

colnames(X(resToy3)) <- clab

rownames(X(resToy3)) <- llab


plot(resToy3,dim=c(1,2),label.tol=0.1,col.group = groups,lab.size=0.6)
plot(resToy3,dim=c(1,3),label.tol=0.1,col.group = groups,lab.size=0.6)
plot(resToy3,dim=c(2,3),label.tol=0.1,col.group = groups,lab.size=0.6)


#----------------------------------------
# DEMO2: Laura van't Veer's gene expression
#        data set for breast cancer
#----------------------------------------


avail <- require(fabiaData)

if (!avail) {
```

```
    message("")
    message("")
    message("#####################################################")
    message("Package 'fabiaData' is not available: please install.")
    message("#####################################################")
} else {

data(Breast_A)

X <- as.matrix(XBreast)


resBreast3 <- fabiap(X,5,0.1,400)

extractPlot(resBreast3,ti="FABIAP Breast cancer(Veer)")

raBreast3 <- extractBic(resBreast3)

if ((raBreast3$bic[[1]][1]>1) && (raBreast3$bic[[1]][2])>1) {
    plotBicluster(raBreast3,1)
}
if ((raBreast3$bic[[2]][1]>1) && (raBreast3$bic[[2]][2])>1) {
    plotBicluster(raBreast3,2)
}
if ((raBreast3$bic[[3]][1]>1) && (raBreast3$bic[[3]][2])>1) {
    plotBicluster(raBreast3,3)
}
if ((raBreast3$bic[[4]][1]>1) && (raBreast3$bic[[4]][2])>1) {
    plotBicluster(raBreast3,4)
}

plot(resBreast3,dim=c(1,2),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast3,dim=c(1,3),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast3,dim=c(1,4),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast3,dim=c(1,5),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast3,dim=c(2,3),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast3,dim=c(2,4),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast3,dim=c(2,5),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast3,dim=c(3,4),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast3,dim=c(3,5),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast3,dim=c(4,5),label.tol=0.03,col.group=CBreast,lab.size=0.6)

}


#----------------------------------
# DEMO3: Su's multiple tissue types
#        gene expression data set
#----------------------------------


avail <- require(fabiaData)
```

```
if (!avail) {
    message("")
    message("")
    message("#####################################################")
    message("Package 'fabiaData' is not available: please install.")
    message("#####################################################")
} else {

data(Multi_A)

X <- as.matrix(XMulti)

resMulti3 <- fabiap(X,5,0.1,300)

extractPlot(resMulti3,ti="FABIAP Multiple tissues(Su)")


raMulti3 <- extractBic(resMulti3)

if ((raMulti3$bic[[1]][1]>1) && (raMulti3$bic[[1]][2])>1) {
    plotBicluster(raMulti3,1)
}
if ((raMulti3$bic[[2]][1]>1) && (raMulti3$bic[[2]][2])>1) {
    plotBicluster(raMulti3,2)
}
if ((raMulti3$bic[[3]][1]>1) && (raMulti3$bic[[3]][2])>1) {
    plotBicluster(raMulti3,3)
}
if ((raMulti3$bic[[4]][1]>1) && (raMulti3$bic[[4]][2])>1) {
    plotBicluster(raMulti3,4)
}

plot(resMulti3,dim=c(1,2),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti3,dim=c(1,3),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti3,dim=c(1,4),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti3,dim=c(1,5),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti3,dim=c(2,3),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti3,dim=c(2,4),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti3,dim=c(2,5),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti3,dim=c(3,4),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti3,dim=c(3,5),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti3,dim=c(4,5),label.tol=0.01,col.group=CMulti,lab.size=0.6)

}


#----------------------------------------
# DEMO4: Rosenwald's diffuse large-B-cell
#        lymphoma gene expression data set
#----------------------------------------

avail <- require(fabiaData)
```

```
if (!avail) {
    message("")
    message("")
    message("#####################################################")
    message("Package 'fabiaData' is not available: please install.")
    message("#####################################################")
} else {


data(DLBCL_B)

X <- as.matrix(XDLBCL)


resDLBCL3 <- fabiap(X,5,0.1,400)

extractPlot(resDLBCL3,ti="FABIAP Lymphoma(Rosenwald)")
raDLBCL3 <- extractBic(resDLBCL3)

if ((raDLBCL3$bic[[1]][1]>1) && (raDLBCL3$bic[[1]][2])>1) {
    plotBicluster(raDLBCL3,1)
}
if ((raDLBCL3$bic[[2]][1]>1) && (raDLBCL3$bic[[2]][2])>1) {
    plotBicluster(raDLBCL3,2)
}
if ((raDLBCL3$bic[[3]][1]>1) && (raDLBCL3$bic[[3]][2])>1) {
    plotBicluster(raDLBCL3,3)
}
if ((raDLBCL3$bic[[4]][1]>1) && (raDLBCL3$bic[[4]][2])>1) {
    plotBicluster(raDLBCL3,4)
}

plot(resDLBCL3,dim=c(1,2),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL3,dim=c(1,3),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL3,dim=c(1,4),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL3,dim=c(1,5),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL3,dim=c(2,3),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL3,dim=c(2,4),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL3,dim=c(2,5),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL3,dim=c(3,4),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL3,dim=c(3,5),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL3,dim=c(4,5),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)


}


## End(Not run)
```

`fabias` *Factor Analysis for Bicluster Acquisition: Sparseness Projection (FABIAS)*

Description

`fabias`: C implementation of `fabias`.

Usage

```
fabias(X,p=13,alpha=0.6,cyc=500,spz=0.5,non_negative=0,random=1.0,center=2,norm=1,lap=1.0,nL=0,lL=0
```

Arguments

`X`	the data matrix.
`p`	number of hidden factors = number of biclusters; default = 13.
`alpha`	sparseness loadings (0.1 - 1.0); default = 0.1.
`cyc`	number of iterations; default = 500.
`spz`	sparseness factors (0.5 - 2.0); default = 0.5 (Laplace).
`non_negative`	Non-negative factors and loadings if non_negative > 0; default = 0.
`random`	<=0: by SVD, >0: random initialization of loadings in [-random,random]; default = 1.0.
`center`	data centering: 1 (mean), 2 (median), > 2 (mode), 0 (no); default = 2.
`norm`	data normalization: 1 (0.75-0.25 quantile), >1 (var=1), 0 (no); default = 1.
`lap`	minimal value of the variational parameter; default = 1.0.
`nL`	maximal number of biclusters at which a row element can participate; default = 0 (no limit)
`lL`	maximal number of row elements per bicluster; default = 0 (no limit)
`bL`	cycle at which the nL or lL maximum starts; default = 0 (start at the beginning)

Details

Biclusters are found by sparse factor analysis where *both* the factors and the loadings are sparse.

Essentially the model is the sum of outer products of vectors:

$$X = \sum_{i=1}^{p} \lambda_i z_i^T + U$$

where the number of summands p is the number of biclusters. The matrix factorization is

$$X = LZ + U$$

Here λ_i are from R^n, z_i from R^l, L from $R^{n \times p}$, Z from $R^{p \times l}$, and X, U from $R^{n \times l}$.

If the nonzero components of the sparse vectors are grouped together then the outer product results in a matrix with a nonzero block and zeros elsewhere.

The model selection is performed by a variational approach according to Girolami 2001 and Palmer et al. 2006.

The prior has finite support, therefore after each update of the loadings they are projected to the finite support. The projection is done according to Hoyer, 2004: given an l_1-norm and an l_2-norm minimize the Euclidean distance to the original vector (currently the l_2-norm is fixed to 1). The projection is a convex quadratic problem which is solved iteratively where at each iteration at least one component is set to zero. Instead of the l_1-norm a sparseness measurement is used which relates the l_1-norm to the l_2-norm.

The code is implemented in C.

Value

object of the class `Factorization`. Containing `LZ` (estimated noise free data LZ), `L` (loadings L), `Z` (factors Z), `U` (noise $X - LZ$), `center` (centering vector), `scaleData` (scaling vector), `X` (centered and scaled data X), `Psi` (noise variance σ), `lapla` (variational parameter), `avini` (the information which the factor z_{ij} contains about x_j averaged over j) `xavini` (the information which the factor z_j contains about x_j) `ini` (for each j the information which the factor z_{ij} contains about x_j).

Author(s)

Sepp Hochreiter

References

S. Hochreiter et al., 'FABIA: Factor Analysis for Bicluster Acquisition', Bioinformatics 26(12):1520-1527, 2010. http://bioinformatics.oxfordjournals.org/cgi/content/abstract/btq227

Mark Girolami, 'A Variational Method for Learning Sparse and Overcomplete Representations', Neural Computation 13(11): 2517-2532, 2001.

J. Palmer, D. Wipf, K. Kreutz-Delgado, B. Rao, 'Variational EM algorithms for non-Gaussian latent variable models', Advances in Neural Information Processing Systems 18, pp. 1059-1066, 2006.

Patrik O. Hoyer, 'Non-negative Matrix Factorization with Sparseness Constraints', Journal of Machine Learning Research 5:1457-1469, 2004.

See Also

`fabia`, `fabias`, `fabiap`, `spfabia`, `fabi`, `fabiasp`, `mfsc`, `nmfdiv`, `nmfeu`, `nmfsc`, `extractPlot`, `extractBic`, `plotBicluster`, `Factorization`, `projFuncPos`, `projFunc`, `estimateMode`, `makeFabiaData`, `makeFabiaDataBlocks`, `makeFabiaDataPos`, `makeFabiaDataBlocksPos`, `matrixImagePlot`, `fabiaDemo`, `fabiaVersion`

Examples

```
#---------------
# TEST
#---------------

dat <- makeFabiaDataBlocks(n = 100,l= 50,p = 3,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

X <- dat[[1]]
Y <- dat[[2]]




resEx <- fabias(X,3,0.6,50)


## Not run:
#-----------------
# DEMO1: Toy Data
#-----------------

n = 1000
l= 100
p = 10

dat <- makeFabiaDataBlocks(n = n,l= l,p = p,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

X <- dat[[1]]
Y <- dat[[2]]
ZC <- dat[[3]]
LC <- dat[[4]]

gclab <- rep.int(0,l)
gllab <- rep.int(0,n)
clab <- as.character(1:l)
llab <- as.character(1:n)
for (i in 1:p){
 for (j in ZC[i]){
     clab[j] <- paste(as.character(i),"_",clab[j],sep="")
 }
 for (j in LC[i]){
     llab[j] <- paste(as.character(i),"_",llab[j],sep="")
 }
 gclab[unlist(ZC[i])] <- gclab[unlist(ZC[i])] + p^i
 gllab[unlist(LC[i])] <- gllab[unlist(LC[i])] + p^i
}
```

```
groups <- gclab

#### FABIAS

resToy2 <- fabias(X,13,0.6,400)

extractPlot(resToy2,ti="FABIAS",Y=Y)


raToy2 <- extractBic(resToy2)

if ((raToy2$bic[[1]][1]>1) && (raToy2$bic[[1]][2])>1) {
    plotBicluster(raToy2,1)
}
if ((raToy2$bic[[2]][1]>1) && (raToy2$bic[[2]][2])>1) {
    plotBicluster(raToy2,2)
}
if ((raToy2$bic[[3]][1]>1) && (raToy2$bic[[3]][2])>1) {
    plotBicluster(raToy2,3)
}
if ((raToy2$bic[[4]][1]>1) && (raToy2$bic[[4]][2])>1) {
    plotBicluster(raToy2,4)
}

colnames(X(resToy2)) <- clab

rownames(X(resToy2)) <- llab


plot(resToy2,dim=c(1,2),label.tol=0.1,col.group = groups,lab.size=0.6)
plot(resToy2,dim=c(1,3),label.tol=0.1,col.group = groups,lab.size=0.6)
plot(resToy2,dim=c(2,3),label.tol=0.1,col.group = groups,lab.size=0.6)


#-----------------------------------------
# DEMO2: Laura van't Veer's gene expression
#        data set for breast cancer
#-----------------------------------------

avail <- require(fabiaData)

if (!avail) {
    message("")
    message("")
    message("#####################################################")
    message("Package 'fabiaData' is not available: please install.")
    message("#####################################################")
} else {


data(Breast_A)

X <- as.matrix(XBreast)
```

```
resBreast2 <- fabias(X,5,0.6,300)

extractPlot(resBreast2,ti="FABIAS Breast cancer(Veer)")


raBreast2 <- extractBic(resBreast2)

if ((raBreast2$bic[[1]][1]>1) && (raBreast2$bic[[1]][2])>1) {
    plotBicluster(raBreast2,1)
}
if ((raBreast2$bic[[2]][1]>1) && (raBreast2$bic[[2]][2])>1) {
    plotBicluster(raBreast2,2)
}
if ((raBreast2$bic[[3]][1]>1) && (raBreast2$bic[[3]][2])>1) {
    plotBicluster(raBreast2,3)
}
if ((raBreast2$bic[[4]][1]>1) && (raBreast2$bic[[4]][2])>1) {
    plotBicluster(raBreast2,4)
}

plot(resBreast2,dim=c(1,2),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast2,dim=c(1,3),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast2,dim=c(1,4),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast2,dim=c(1,5),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast2,dim=c(2,3),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast2,dim=c(2,4),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast2,dim=c(2,5),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast2,dim=c(3,4),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast2,dim=c(3,5),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast2,dim=c(4,5),label.tol=0.03,col.group=CBreast,lab.size=0.6)

}

#-----------------------------------
# DEMO3: Su's multiple tissue types
#        gene expression data set
#-----------------------------------

avail <- require(fabiaData)

if (!avail) {
    message("")
    message("")
    message("########################################################")
    message("Package 'fabiaData' is not available: please install.")
    message("########################################################")
} else {

data(Multi_A)

X <- as.matrix(XMulti)
```

```
resMulti2 <- fabias(X,5,0.6,300)

extractPlot(resMulti2,ti="FABIAS Multiple tissues(Su)")

raMulti2 <- extractBic(resMulti2)

if ((raMulti2$bic[[1]][1]>1) && (raMulti2$bic[[1]][2])>1) {
    plotBicluster(raMulti2,1)
}
if ((raMulti2$bic[[2]][1]>1) && (raMulti2$bic[[2]][2])>1) {
    plotBicluster(raMulti2,2)
}
if ((raMulti2$bic[[3]][1]>1) && (raMulti2$bic[[3]][2])>1) {
    plotBicluster(raMulti2,3)
}
if ((raMulti2$bic[[4]][1]>1) && (raMulti2$bic[[4]][2])>1) {
    plotBicluster(raMulti2,4)
}

plot(resMulti2,dim=c(1,2),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti2,dim=c(1,3),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti2,dim=c(1,4),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti2,dim=c(1,5),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti2,dim=c(2,3),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti2,dim=c(2,4),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti2,dim=c(2,5),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti2,dim=c(3,4),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti2,dim=c(3,5),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti2,dim=c(4,5),label.tol=0.01,col.group=CMulti,lab.size=0.6)

}


#-----------------------------------------
# DEMO4: Rosenwald's diffuse large-B-cell
#        lymphoma gene expression data set
#-----------------------------------------


avail <- require(fabiaData)

if (!avail) {
    message("")
    message("")
    message("#######################################################")
    message("Package 'fabiaData' is not available: please install.")
    message("#######################################################")
} else {

data(DLBCL_B)

X <- as.matrix(XDLBCL)
```

```
resDLBCL2 <- fabias(X,5,0.6,300)

extractPlot(resDLBCL2,ti="FABIAS Lymphoma(Rosenwald)")

raDLBCL2 <- extractBic(resDLBCL2)

if ((raDLBCL2$bic[[1]][1]>1) && (raDLBCL2$bic[[1]][2])>1) {
    plotBicluster(raDLBCL2,1)
}
if ((raDLBCL2$bic[[2]][1]>1) && (raDLBCL2$bic[[2]][2])>1) {
    plotBicluster(raDLBCL2,2)
}
if ((raDLBCL2$bic[[3]][1]>1) && (raDLBCL2$bic[[3]][2])>1) {
    plotBicluster(raDLBCL2,3)
}
if ((raDLBCL2$bic[[4]][1]>1) && (raDLBCL2$bic[[4]][2])>1) {
    plotBicluster(raDLBCL2,4)
}

plot(resDLBCL2,dim=c(1,2),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL2,dim=c(1,3),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL2,dim=c(1,4),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL2,dim=c(1,5),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL2,dim=c(2,3),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL2,dim=c(2,4),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL2,dim=c(2,5),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL2,dim=c(3,4),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL2,dim=c(3,5),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL2,dim=c(4,5),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)

}



## End(Not run)
```

`fabiasp` *Factor Analysis for Bicluster Acquisition: Sparseness Projection (FABIASP)*

Description

`fabiasp`: R implementation of `fabias`, therefore it is **slow**.

Usage

```
fabiasp(X,p=13,alpha=0.6,cyc=500,spz=0.5,center=2,norm=1,lap=1.0)
```

Arguments

X	the data matrix.
p	number of hidden factors = number of biclusters; default = 13.
alpha	sparseness loadings (0.1 - 1.0); default = 0.6.
cyc	number of iterations; default = 500.
spz	sparseness factors (0.5 - 2.0); default = 0.5 (Laplace).
center	data centering: 1 (mean), 2 (median), > 2 (mode), 0 (no); default = 2.
norm	data normalization: 1 (0.75-0.25 quantile), >1 (var=1), 0 (no); default = 1.
lap	minimal value of the variational parameter; default = 1.0.

Details

Biclusters are found by sparse factor analysis where *both* the factors and the loadings are sparse.

Essentially the model is the sum of outer products of vectors:

$$X = \sum_{i=1}^{p} \lambda_i z_i^T + U$$

where the number of summands p is the number of biclusters. The matrix factorization is

$$X = LZ + U$$

Here λ_i are from R^n, z_i from R^l, L from $R^{n \times p}$, Z from $R^{p \times l}$, and X, U from $R^{n \times l}$.

If the nonzero components of the sparse vectors are grouped together then the outer product results in a matrix with a nonzero block and zeros elsewhere.

The model selection is performed by a variational approach according to Girolami 2001 and Palmer et al. 2006.

The prior has finite support, therefore after each update of the loadings they are projected to the finite support. The projection is done according to Hoyer, 2004: given an l_1-norm and an l_2-norm minimize the Euclidean distance to the original vector (currently the l_2-norm is fixed to 1). The projection is a convex quadratic problem which is solved iteratively where at each iteration at least one component is set to zero. Instead of the l_1-norm a sparseness measurement is used which relates the l_1-norm to the l_2-norm.

The code is implemented in R, therefore it is **slow**.

Value

object of the class `Factorization`. Containing `LZ` (estimated noise free data LZ), `L` (loadings L), `Z` (factors Z), `U` (noise $X - LZ$), `center` (centering vector), `scaleData` (scaling vector), `X` (centered and scaled data X), `Psi` (noise variance σ), `lapla` (variational parameter), `avini` (the information which the factor z_{ij} contains about x_j averaged over j) `xavini` (the information which the factor z_j contains about x_j) `ini` (for each j the information which the factor z_{ij} contains about x_j).

Author(s)

Sepp Hochreiter

References

S. Hochreiter et al., 'FABIA: Factor Analysis for Bicluster Acquisition', Bioinformatics 26(12):1520-1527, 2010. http://bioinformatics.oxfordjournals.org/cgi/content/abstract/btq227

Mark Girolami, 'A Variational Method for Learning Sparse and Overcomplete Representations', Neural Computation 13(11): 2517-2532, 2001.

J. Palmer, D. Wipf, K. Kreutz-Delgado, B. Rao, 'Variational EM algorithms for non-Gaussian latent variable models', Advances in Neural Information Processing Systems 18, pp. 1059-1066, 2006.

Patrik O. Hoyer, 'Non-negative Matrix Factorization with Sparseness Constraints', Journal of Machine Learning Research 5:1457-1469, 2004.

See Also

fabia, fabias, fabiap, spfabia, fabi, fabiasp, mfsc, nmfdiv, nmfeu, nmfsc, extractPlot, extractBic, plotBicluster, Factorization, projFuncPos, projFunc, estimateMode, makeFabiaData, makeFabiaDataBlocks, makeFabiaDataPos, makeFabiaDataBlocksPos, matrixImagePlot, fabiaDemo, fabiaVersion

Examples

```
#---------------
# TEST
#---------------

dat <- makeFabiaDataBlocks(n = 100,l= 50,p = 3,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

X <- dat[[1]]
Y <- dat[[2]]



resEx <- fabiasp(X,3,0.6,50)


## Not run:
#---------------
# DEMO1
#---------------

dat <- makeFabiaDataBlocks(n = 1000,l= 100,p = 10,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

X <- dat[[1]]
Y <- dat[[2]]
```

```
resToy <- fabiasp(X,13,0.6,200)

extractPlot(resToy,"ti=FABIASP",Y=Y)

#---------------
# DEMO2
#---------------


avail <- require(fabiaData)

if (!avail) {
    message("")
    message("")
    message("#######################################################")
    message("Package 'fabiaData' is not available: please install.")
    message("#######################################################")
} else {


data(Breast_A)

X <- as.matrix(XBreast)

resBreast <- fabiasp(X,5,0.6,200)

extractPlot(resBreast,ti="FABIASP Breast cancer(Veer)")

#sorting of predefined labels
CBreast

}

#---------------
# DEMO3
#---------------


avail <- require(fabiaData)

if (!avail) {
    message("")
    message("")
    message("#######################################################")
    message("Package 'fabiaData' is not available: please install.")
    message("#######################################################")
} else {

data(Multi_A)

X <- as.matrix(XMulti)
```

```
resMulti <- fabiasp(X,5,0.6,200)

extractPlot(resMulti,"ti=FABIASP Multiple tissues(Su)")

#sorting of predefined labels
CMulti

}


#---------------
# DEMO4
#---------------


avail <- require(fabiaData)

if (!avail) {
    message("")
    message("")
    message("#######################################################")
    message("Package 'fabiaData' is not available: please install.")
    message("#######################################################")
} else {

data(DLBCL_B)

X <- as.matrix(XDLBCL)


resDLBCL <- fabiasp(X,5,0.6,200)

extractPlot(resDLBCL,ti="FABIASP Lymphoma(Rosenwald)")

#sorting of predefined labels
CDLBCL
}



## End(Not run)
```

`fabiaVersion` *Display version info for package and for FABIA*

Description

`fabiaVersion` displays version information about the package.

Usage

```
fabiaVersion()
```

Value

Displays version information

Author(s)

Sepp Hochreiter

See Also

fabia, fabias, fabiap, spfabia, readSpfabiaResult, fabi, fabiasp, mfsc, nmfdiv, nmfeu, nmfsc, extractPlot, extractBic, plotBicluster, Factorization, projFuncPos, projFunc, estimateMode, makeFabiaData, makeFabiaDataBlocks, makeFabiaDataPos, makeFabiaDataBlocksPos, matrixImagePlot, fabiaDemo, fabiaVersion

Examples

```
fabiaVersion()
```

Factorization-class *Factorization instances*

Description

Factorization is a class to store results of matrix factorization algorithms. It has been designed for biclustering but can be used for "principal component analysis", "singular value decomposition", "independent component analysis", "factor analysis", and "non-negative matrix factorization".

Usage

```
## S4 method for signature 'Factorization'
plot(x, Rm=NULL, Cm=NULL, dim = c(1, 2),
    zoom = rep(1, 2), col.group = NULL,
    colors = c("orange1", "red", rainbow(length(unique(col.group)),
               start=2/6, end=4/6)),
    col.areas = TRUE, col.symbols = c(1, rep(2, length(unique(col.group)))),
    sampleNames = TRUE, rot = rep(-1, length(dim)),
    labels = NULL, label.tol = 0.1, lab.size = 0.725, col.size = 10,
    row.size = 10, do.smoothScatter = FALSE,
    do.plot = TRUE, ... )

## S4 method for signature 'Factorization'
show(object)
```

```
## S4 method for signature 'Factorization'
showSelected(object, which=c(1,2,3,4))

## S4 method for signature 'Factorization'
summary(object, ...)
```

Arguments

	PLOT:
x	object of the class Factorization.
Rm	row weighting vector. If NULL, it defaults to rep(1,nrow(L(x))).
Cm	column weighting vector. If NULL, it defaults to rep(1,ncol(Z(x))).
dim	optional principal factors that are plotted along the horizontal and vertical axis. Defaults to c(1,2).
zoom	optional zoom factor for row and column items. Defaults to c(1,1).
col.group	optional vector (character or numeric) indicating the different groupings of the columns. Defaults to 1.
colors	vector specifying the colors for the annotation of the plot; the first two elements concern the rows; the third till the last element concern the columns; the first element will be used to color the unlabeled rows; the second element for the labeled rows and the remaining elements to give different colors to different groups of columns. Defaults to c("orange1", "red", rainbow(length(unique(col.group)), start=2/6, end=4/6)).
col.areas	logical value indicating whether columns should be plotted as squares with areas proportional to their marginal mean and colors representing the different groups (TRUE), or with symbols representing the groupings and identical size (FALSE). Defaults to TRUE.
col.symbols	vector of symbols when col.areas=FALSE corresponds to the pch argument of the function plot. Defaults to c(1, rep(2, length(unique(col.group)))).
sampleNames	either a logical vector of length one or a character vector of length equal to the number of samples in the dataset. If a logical is provided, sample names will be displayed on the plot (TRUE; default) or not (FALSE); if a character vector is provided, the names provided will be used to label the samples instead of the default column names.
rot	rotation of plot. Defaults to c(-1,-1).
labels	character vector to be used for labeling points on the graph; if NULL (default), the row names of x are used instead.
label.tol	numerical value specifying either the percentile (label.tol<=1) of rows or the number of rows (label.tol>1) most distant from the plot-center (0,0) that are labeled and are plotted as circles with area proportional to the marginal means of the original data. Defaults to 1.
lab.size	size of identifying labels for row- and column-items as cex parameter of the text function. Defaults to 0.725.

`col.size` size of the column symbols in mm. Defaults to `10`.

`row.size` size of the row symbols in mm. Defaults to `10`.

`do.smoothScatter` use smoothScatter or not instead of plotting individual points. Defaults to `FALSE`.

`do.plot` produce a plot or not. Defaults to `TRUE`.

`...` further arguments are passed on to `eqscaleplotLoc` which draws the canvas for the plot; useful for adding a `main` or a custom `sub`.

SHOW:

`object` An instance of `Factorization-class` .

SHOWSELECTED:

see `object` at `show`.

`which` used to provide a list of which plots should be generated: 1=the information content of biclusters, 2=the information content of samples, 3=the loadings per bicluster, 4=the factors per bicluster, default c(1,2,3,4).

SUMMARY:

see `object` at `show`.

... further arguments.

Details

Plot Produces a biplot of a matrix factorization result stored in an instance of the Factorization class.

The function `plot` is based on the function `plot.mpm` in the R package `mpm` (Version: 1.0-16, Date: 2009-08-26, Title: Multivariate Projection Methods, Maintainer: Tobias Verbeke <tobias.verbeke@openanalytics.be>, Author: Luc Wouters <wouters_luc@telenet.be>).

Biclusters are found by sparse factor analysis where *both* the factors and the loadings are sparse.

Essentially the model is the sum of outer products of vectors:

$$X = \sum_{i=1}^{p} \lambda_i z_i^T + U$$

where the number of summands p is the number of biclusters. The matrix factorization is

$$X = LZ + U$$

Here λ_i are from R^n, z_i from R^l, L from $R^{n \times p}$, Z from $R^{p \times l}$, and X, U from $R^{n \times l}$.

For noise free projection like independent component analysis we set the noise term to zero: $U = 0$.

The argument `label.tol` can be used to select the most informative rows, i.e. rows that are most distant from the center of the plot (smaller 1: percentage of rows, larger 1: number of rows).

Only these row-items are then labeled and represented as circles with their areas proportional to the row weighting.

If the column-items are grouped these groups can be visualized by colors given by `col.group`.

Show Statistics of a matrix factorization result stored in an instance of the Factorization class.

This function supplies statistics on a matrix factorization result which is stored as an instance of Factorization-class.

The following is plotted:

1. the information content of biclusters.
2. the information content of samples.
3. the loadings per bicluster.
4. the factors per bicluster.

ShowSelected Lists selected statistics of a matrix factorization result stored in an instance of the Factorization class.

This function supplies selected statistics on a matrix factorization result which is stored as an instance of Factorization-class.

The following is plotted depending on the display selection variable `which`:

1. the information content of biclusters.
2. the information content of samples.
3. the loadings per bicluster.
4. the factors per bicluster.

Summary Summary of matrix factorization result stored in an instance of the Factorization class.

This function gives information on a matrix factorization result which is stored as an instance of Factorization-class.

The summary consists of following items:

1. the number or rows and columns of the original matrix.
2. the number of clusters for rows and columns is given.
3. for the row cluster the information content is given.
4. for each column its information is given.
5. for each column cluster a summary is given.
6. for each row cluster a summary is given.

Value

FACTORIZATION:

An instance of Factorization-class .

PLOT:

`Rows` a list with the X and Y coordinates of the rows and an indication `Select` of whether the row was selected according to `label.tol`.

`Columns` a list with the X and Y coordinates of the columns.

SHOW:

no value.

SHOWSELECTED:

no value.

SUMMARY:

no value.

Slots

Objects of class `Factorization` have the following slots:

`parameters`: Saves parameters of the factorization method in a list: ("method","number of cycles","sparseness weight","sparseness prior for loadings","sparseness prior for factors","number biclusters","projection sparseness loadings", "projection sparseness factors","initialization range","are loadings rescaled after each iterations","normalization = scaling of rows","centering method of rows","parameter for method").

`n`: number of rows, left dimension.

`p1`: right dimension of left matrix.

`p2`: left dimension of right matrix.

`l`: number of columns, right dimension.

`center`: vector of the centers.

`scaleData`: vector of the scaling factors.

`X`: centered and scaled data matrix n x l.

`L`: left matrix n x p1.

`Z`: right matrix p2 x l.

`M`: middle matrix p1 x p2.

`LZ`: matrix L x M x Z.

`U`: noise matrix.

`avini`: information of each bicluster, vector of length p2.

`xavini`: information extracted from each sample, vector of length l.

`ini`: information of each bicluster in each sample, matrix p2 x l.

`Psi`: noise variance per row, vector of length n.

`lapla`: prior information for each sample, vector of length l.

Constructor

Constructor of class Factorization.

```
Factorization(parameters=list(),n=1,p1=1,p2=1,l=1,center=as.vector(1),scaleData=as.vector(1),X=as.m
```

Accessors

In the following `x` denotes a Factorization object.

`parameters(x)`, `parameters(x) <- value`: Returns or sets `parameters`, where the return value and `value` are both an instance of `list`. Parameters of the factorization method are stored in a list: ("method","number of cycles","sparseness weight","sparseness prior for loadings","sparseness prior for factors","number biclusters","projection sparseness loadings", "projection sparseness factors","initialization range","are loadings rescaled after each iterations","normalization = scaling of rows","centering method of rows","parameter for method").

`n(x)`, `n(x) <- value`: Returns or sets `n`, where the return value and `value` are both an instance of `numeric`. Number of rows, left dimension.

`p1(x)`, `p1(x) <- value`: Returns or sets `p1`, where the return value and `value` are both an instance of `numeric`. Right dimension of left matrix

`p2(x)`, `p2(x) <- value`: Returns or sets `p2`, where the return value and `value` are both an instance of `numeric`. Left dimension of right matrix.

`l(x)`, `l(x) <- value`: Returns or sets `l`, where the return value and `value` are both an instance of `numeric`. Number of columns, right dimension.

`center(x)`, `center(x) <- value`: Returns or sets `center`, where the return value and `value` are both an instance of `numeric`. Vector of the centers.

`scaleData(x)`, `scaleData(x) <- value`: Returns or sets `scaleData`, where the return value and `value` are both an instance of `numeric`. Vector of the scaling factors.

`X(x)`, `X(x) <- value`: Returns or sets `X`, where the return value and `value` are both an instance of `matrix`. Centered and scaled data matrix n x l.

`L(x)`, `L(x) <- value`: Returns or sets `L`, where the return value and `value` are both an instance of `matrix`. Left matrix n x p1.

`Z(x)`, `Z(x) <- value`: Returns or sets `Z`, where the return value and `value` are both an instance of `matrix`. Right matrix p2 x l.

`M(x)`, `M(x) <- value`: Returns or sets `M`, where the return value and `value` are both an instance of `matrix`. Middle matrix p1 x p2.

`LZ(x)`, `LZ(x) <- value`: Returns or sets `LZ`, where the return value and `value` are both an instance of `matrix`. Matrix L x M x Z.

`U(x)`, `U(x) <- value`: Returns or sets `U`, where the return value and `value` are both an instance of `matrix`. Noise matrix.

`avini(x)`, `avini(x) <- value`: Returns or sets `avini`, where the return value and `value` are both an instance of `numeric`. Information of each bicluster, vector of length p2.

`xavini(x)`, `xavini(x) <- value`: Returns or sets `xavini`, where the return value and `value` are both an instance of `numeric`. Information extracted from each sample, vector of length l.

`ini(x)`, `ini(x) <- value`: Returns or sets `ini`, where the return value and `value` are both an instance of `matrix`. Information of each bicluster in each sample, matrix p2 x l.

`Psi(x)`, `Psi(x) <- value`: Returns or sets `Psi`, where the return value and `value` are both an instance of `numeric`. Noise variance per row, vector of length n.

`lapla(x)`, `lapla(x) <- value`: Returns or sets `lapla`, where the return value and `value` are both an instance of `matrix`. Prior information for each sample, vector of length l.

Signatures

plot `signature(x = "Factorization", y = "missing")` Plot of a matrix factorization result

show `signature(object = "Factorization")` Display statistics of a matrix factorization result

showSelected `signature(object = "Factorization", which = "numeric")` Display particular statistics of a matrix factorization result

summary `signature(object = "Factorization")` Summary of matrix factorization result

Functions that return objects of this class

Factorization objects are returned by `fabia`, `fabias`, `fabiap`, `fabiasp`, `mfsc`, `nmfsc`, `nmfdiv`, and `nmfeu`.

Extension to store results of other methods

The class `Factorization` may contain the result of different matrix factorization methods. The methods may be generative or not.

Methods my be "singular value decomposition" (M contains singular values as well as avini, L and Z are orthonormal matrices), "independent component analysis" (Z contains the projection/sources, L is the mixing matrix, M is unity), "factor analysis" (Z contains factors, L the loadings, M is unity, U the noise, Psi the noise covariance, lapla is a variational parameter for non-Gaussian factors, avini and ini are the information the factors convey about the observations).

Author(s)

Sepp Hochreiter

See Also

`fabia`, `fabias`, `fabiap`, `fabi`, `fabiasp`, `mfsc`, `nmfdiv`, `nmfeu`, `nmfsc`, `extractPlot`, `extractBic`, `plotBicluster`, `Factorization`, `projFuncPos`, `projFunc`, `estimateMode`, `makeFabiaData`, `makeFabiaDataBlocks`, `makeFabiaDataPos`, `makeFabiaDataBlocksPos`, `matrixImagePlot`, `fabiaDemo`, `fabiaVersion`

Examples

```
###################
# TEST
###################


#------------------
#   PLOT
#------------------



n=200
l=100
p=4

dat <- makeFabiaDataBlocks(n = n,l= l,p = p,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

X <- dat[[1]]
ZC <- dat[[3]]
LC <- dat[[4]]


resEx <- fabia(X,p,0.01,400)


gclab <- rep.int(0,l)
gllab <- rep.int(0,n)
clab <- as.character(1:l)
```

```
llab <- as.character(1:n)
for (i in 1:p){
 for (j in ZC[i]){
     clab[j] <- paste(as.character(i),"_",clab[j],sep="")
 }
 for (j in LC[i]){
     llab[j] <- paste(as.character(i),"_",llab[j],sep="")
 }
 gclab[unlist(ZC[i])] <- gclab[unlist(ZC[i])] + p^i
 gllab[unlist(LC[i])] <- gllab[unlist(LC[i])] + p^i
}


groups <- gclab

colnames(X(resEx)) <- clab

rownames(X(resEx)) <- llab


plot(resEx,dim=c(1,2),label.tol=0.1,col.group = groups,lab.size=0.6)
plot(resEx,dim=c(1,3),label.tol=0.1,col.group = groups,lab.size=0.6)
plot(resEx,dim=c(2,3),label.tol=0.1,col.group = groups,lab.size=0.6)



#------------------
#   SHOW
#------------------


dat <- makeFabiaDataBlocks(n = 100,l= 50,p = 3,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

X <- dat[[1]]


resEx <- fabia(X,3,0.01,100)

show(resEx)



#------------------
# SHOWSELECTED
#------------------

dat <- makeFabiaDataBlocks(n = 100,l= 50,p = 3,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

X <- dat[[1]]
```

```
resEx <- fabia(X,3,0.01,100)

showSelected(resEx,which=1)
showSelected(resEx,which=2)



#------------------
# SUMMARY
#------------------

dat <- makeFabiaDataBlocks(n = 100,l= 50,p = 3,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

X <- dat[[1]]


resEx <- fabia(X,3,0.01,100)

summary(resEx)
```

`makeFabiaData` *Generation of Bicluster Data*

Description

`makeFabiaData`: R implementation of `makeFabiaData`.

Usage

```
makeFabiaData(n,l,p,f1,f2,of1,of2,sd_noise,sd_z_noise,
              mean_z,sd_z,sd_l_noise,mean_l,sd_l)
```

Arguments

`n`	number of observations.
`l`	number of samples.
`p`	number of biclusters.
`f1`	nn/f1 max. additional samples are active in a bicluster.
`f2`	n/f2 max. additional observations that form a pattern in a bicluster.
`of1`	minimal active samples in a bicluster.

of2	minimal observations that form a pattern in a bicluster.
sd_noise	Gaussian zero mean noise std on data matrix.
sd_z_noise	Gaussian zero mean noise std for deactivated hidden factors.
mean_z	Gaussian mean for activated factors.
sd_z	Gaussian std for activated factors.
sd_l_noise	Gaussian zero mean noise std if no observation patterns are present.
mean_l	Gaussian mean for observation patterns.
sd_l	Gaussian std for observation patterns.

Details

Essentially the data generation model is the sum of outer products of sparse vectors:

$$X = \sum_{i=1}^{p} \lambda_i z_i^T + U$$

where the number of summands p is the number of biclusters. The matrix factorization is

$$X = LZ + U$$

and noise free

$$Y = LZ$$

Here λ_i are from R^n, z_i from R^l, L from $R^{n \times p}$, Z from $R^{p \times l}$, and X, U, Y from $R^{n \times l}$.

Sequentially L_i are generated using n, f2, of2, sd_l_noise, mean_l, sd_l. of2 gives the minimal observations participating in a bicluster to which between 0 and $n/f2$ observations are added, where the number is uniformly chosen. sd_l_noise gives the noise of observations not participating in the bicluster. mean_l and sd_l determines the Gaussian from which the values are drawn for the observations that participate in the bicluster. The sign of the mean is randomly chosen for each component.

Sequentially Z_i are generated using l, f1, of1, sd_z_noise, mean_z, sd_z. of1 gives the minimal samples participating in a bicluster to which between 0 and $l/f1$ samples are added, where the number is uniformly chosen. sd_z_noise gives the noise of samples not participating in the bicluster. mean_z and sd_z determines the Gaussian from which the values are drawn for the samples that participate in the bicluster.

U is the overall Gaussian zero mean noise generated by sd_noise.

Implementation in R.

Value

X	the noise data from $R^{n \times l}$.
Y	the noise free data from $R^{n \times l}$.
ZC	list where i-th element gives samples belonging to i-th bicluster.
LC	list where i-th element gives observations belonging to i-th bicluster.

Author(s)

Sepp Hochreiter

See Also

fabia, fabias, fabiap, fabi, fabiasp, mfsc, nmfdiv, nmfeu, nmfsc, extractPlot, extractBic, plotBicluster, Factorization, projFuncPos, projFunc, estimateMode, makeFabiaData, makeFabiaDataBlocks, makeFabiaDataPos, makeFabiaDataBlocksPos, matrixImagePlot, fabiaDemo, fabiaVersion

Examples

```
#---------------
# TEST
#---------------

dat <- makeFabiaData(n = 100,l= 50,p = 3,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

X <- dat[[1]]
Y <- dat[[2]]

matrixImagePlot(Y)
dev.new()
matrixImagePlot(X)


## Not run:
#---------------
# DEMO
#---------------

dat <- makeFabiaData(n = 1000,l= 100,p = 10,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

X <- dat[[1]]
Y <- dat[[2]]

matrixImagePlot(Y)
dev.new()
matrixImagePlot(X)


## End(Not run)
```

`makeFabiaDataBlocks` *Generation of Bicluster Data with Bicluster Blocks*

Description

`makeFabiaDataBlocks`: R implementation of `makeFabiaDataBlocks`.

Usage

```
makeFabiaDataBlocks(n,l,p,f1,f2,of1,of2,sd_noise,sd_z_noise,
              mean_z,sd_z,sd_l_noise,mean_l,sd_l)
```

Arguments

`n`	number of observations.
`l`	number of samples.
`p`	number of biclusters.
`f1`	nn/f1 max. additional samples are active in a bicluster.
`f2`	n/f2 max. additional observations that form a pattern in a bicluster.
`of1`	minimal active samples in a bicluster.
`of2`	minimal observations that form a pattern in a bicluster.
`sd_noise`	Gaussian zero mean noise std on data matrix.
`sd_z_noise`	Gaussian zero mean noise std for deactivated hidden factors.
`mean_z`	Gaussian mean for activated factors.
`sd_z`	Gaussian std for activated factors.
`sd_l_noise`	Gaussian zero mean noise std if no observation patterns are present.
`mean_l`	Gaussian mean for observation patterns.
`sd_l`	Gaussian std for observation patterns.

Details

Bicluster data is generated for visualization because the biclusters are now in block format. That means observations and samples that belong to a bicluster are consecutive. This allows visual inspection because the use can identify blocks and whether they have been found or reconstructed.

Essentially the data generation model is the sum of outer products of sparse vectors:

$$X = \sum_{i=1}^{p} \lambda_i z_i^T + U$$

where the number of summands p is the number of biclusters. The matrix factorization is

$$X = LZ + U$$

and noise free

$$Y = LZ$$

Here λ_i are from R^n, z_i from R^l, L from $R^{n \times p}$, Z from $R^{p \times l}$, and X, U, Y from $R^{n \times l}$.

Sequentially L_i are generated using `n`, `f2`, `of2`, `sd_l_noise`, `mean_l`, `sd_l`. `of2` gives the minimal observations participating in a bicluster to which between 0 and $n/f2$ observations are added, where the number is uniformly chosen. `sd_l_noise` gives the noise of observations not participating in the bicluster. `mean_l` and `sd_l` determines the Gaussian from which the values are drawn for the observations that participate in the bicluster. The sign of the mean is randomly chosen for each component.

Sequentially Z_i are generated using `l`, `f1`, `of1`, `sd_z_noise`, `mean_z`, `sd_z`. `of1` gives the minimal samples participating in a bicluster to which between 0 and $l/f1$ samples are added, where the number is uniformly chosen. `sd_z_noise` gives the noise of samples not participating in the bicluster. `mean_z` and `sd_z` determines the Gaussian from which the values are drawn for the samples that participate in the bicluster.

U is the overall Gaussian zero mean noise generated by `sd_noise`.

Implementation in R.

Value

`Y`	the noise data from $R^{n \times l}$.
`X`	the noise free data from $R^{n \times l}$.
`ZC`	list where i-th element gives samples belonging to i-th bicluster.
`LC`	list where i-th element gives observations belonging to i-th bicluster.

Author(s)

Sepp Hochreiter

See Also

`fabia`, `fabias`, `fabiap`, `fabi`, `fabiasp`, `mfsc`, `nmfdiv`, `nmfeu`, `nmfsc`, `extractPlot`, `extractBic`, `plotBicluster`, `Factorization`, `projFuncPos`, `projFunc`, `estimateMode`, `makeFabiaData`, `makeFabiaDataBlocks`, `makeFabiaDataPos`, `makeFabiaDataBlocksPos`, `matrixImagePlot`, `fabiaDemo`, `fabiaVersion`

Examples

```
#---------------
# TEST
#---------------

dat <- makeFabiaDataBlocks(n = 100,l= 50,p = 3,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

X <- dat[[1]]
Y <- dat[[2]]
```

```
matrixImagePlot(Y)
dev.new()
matrixImagePlot(X)


## Not run:
#---------------
# DEMO
#---------------

dat <- makeFabiaDataBlocks(n = 1000,l= 100,p = 10,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

Y <- dat[[1]]
X <- dat[[2]]

matrixImagePlot(Y)
dev.new()
matrixImagePlot(X)


## End(Not run)
```

`makeFabiaDataBlocksPos` *Generation of Bicluster Data with Bicluster Blocks*

Description

`makeFabiaDataBlocksPos`: R implementation of `makeFabiaDataBlocksPos`.

Usage

```
makeFabiaDataBlocksPos(n,l,p,f1,f2,of1,of2,sd_noise,sd_z_noise,
              mean_z,sd_z,sd_l_noise,mean_l,sd_l)
```

Arguments

`n`	number of observations.
`l`	number of samples.
`p`	number of biclusters.
`f1`	nn/f1 max. additional samples are active in a bicluster.
`f2`	n/f2 max. additional observations that form a pattern in a bicluster.
`of1`	minimal active samples in a bicluster.

of2	minimal observations that form a pattern in a bicluster.
sd_noise	Gaussian zero mean noise std on data matrix.
sd_z_noise	Gaussian zero mean noise std for deactivated hidden factors.
mean_z	Gaussian mean for activated factors.
sd_z	Gaussian std for activated factors.
sd_l_noise	Gaussian zero mean noise std if no observation patterns are present.
mean_l	Gaussian mean for observation patterns.
sd_l	Gaussian std for observation patterns.

Details

Bicluster data is generated for visualization because the biclusters are now in block format. That means observations and samples that belong to a bicluster are consecutive. This allows visual inspection because the use can identify blocks and whether they have been found or reconstructed.

Essentially the data generation model is the sum of outer products of sparse vectors:

$$X = \sum_{i=1}^{p} \lambda_i z_i^T + U$$

where the number of summands p is the number of biclusters. The matrix factorization is

$$X = LZ + U$$

and noise free

$$Y = LZ$$

Here λ_i are from R^n, z_i from R^l, L from $R^{n \times p}$, Z from $R^{p \times l}$, and X, U, Y from $R^{n \times l}$.

Sequentially L_i are generated using n, f2, of2, sd_l_noise, mean_l, sd_l. of2 gives the minimal observations participating in a bicluster to which between 0 and $n/f2$ observations are added, where the number is uniformly chosen. sd_l_noise gives the noise of observations not participating in the bicluster. mean_l and sd_l determines the Gaussian from which the values are drawn for the observations that participate in the bicluster. "POS": The sign of the mean is fixed.

Sequentially Z_i are generated using l, f1, of1, sd_z_noise, mean_z, sd_z. of1 gives the minimal samples participating in a bicluster to which between 0 and $l/f1$ samples are added, where the number is uniformly chosen. sd_z_noise gives the noise of samples not participating in the bicluster. mean_z and sd_z determines the Gaussian from which the values are drawn for the samples that participate in the bicluster.

U is the overall Gaussian zero mean noise generated by sd_noise.

Implementation in R.

Value

Y	the noise data from $R^{n \times l}$.
X	the noise free data from $R^{n \times l}$.
ZC	list where i-th element gives samples belonging to i-th bicluster.
LC	list where i-th element gives observations belonging to i-th bicluster.

Author(s)

Sepp Hochreiter

See Also

fabia, fabias, fabiap, fabi, fabiasp, mfsc, nmfdiv, nmfeu, nmfsc, extractPlot, extractBic, plotBicluster, Factorization, projFuncPos, projFunc, estimateMode, makeFabiaData, makeFabiaDataBlocks, makeFabiaDataPos, makeFabiaDataBlocksPos, matrixImagePlot, fabiaDemo, fabiaVersion

Examples

```
#---------------
# TEST
#---------------

dat <- makeFabiaDataBlocksPos(n = 100,l= 50,p = 3,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

X <- dat[[1]]
Y <- dat[[2]]

matrixImagePlot(Y)
dev.new()
matrixImagePlot(X)


## Not run:
#---------------
# DEMO
#---------------

dat <- makeFabiaDataBlocksPos(n = 1000,l= 100,p = 10,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

Y <- dat[[1]]
X <- dat[[2]]

matrixImagePlot(Y)
dev.new()
matrixImagePlot(X)


## End(Not run)
```

makeFabiaDataPos *Generation of Bicluster Data*

Description

makeFabiaDataPos: R implementation of makeFabiaDataPos.

Usage

```
makeFabiaDataPos(n,l,p,f1,f2,of1,of2,sd_noise,sd_z_noise,
              mean_z,sd_z,sd_l_noise,mean_l,sd_l)
```

Arguments

n	number of observations.
l	number of samples.
p	number of biclusters.
f1	nn/f1 max. additional samples are active in a bicluster.
f2	n/f2 max. additional observations that form a pattern in a bicluster.
of1	minimal active samples in a bicluster.
of2	minimal observations that form a pattern in a bicluster.
sd_noise	Gaussian zero mean noise std on data matrix.
sd_z_noise	Gaussian zero mean noise std for deactivated hidden factors.
mean_z	Gaussian mean for activated factors.
sd_z	Gaussian std for activated factors.
sd_l_noise	Gaussian zero mean noise std if no observation patterns are present.
mean_l	Gaussian mean for observation patterns.
sd_l	Gaussian std for observation patterns.

Details

Essentially the data generation model is the sum of outer products of sparse vectors:

$$X = \sum_{i=1}^{p} \lambda_i z_i^T + U$$

where the number of summands p is the number of biclusters. The matrix factorization is

$$X = LZ + U$$

and noise free

$$Y = LZ$$

Here λ_i are from R^n, z_i from R^l, L from $R^{n \times p}$, Z from $R^{p \times l}$, and X, U, Y from $R^{n \times l}$.

Sequentially L_i are generated using `n`, `f2`, `of2`, `sd_l_noise`, `mean_l`, `sd_l`. `of2` gives the minimal observations participating in a bicluster to which between 0 and $n/f2$ observations are added, where the number is uniformly chosen. `sd_l_noise` gives the noise of observations not participating in the bicluster. `mean_l` and `sd_l` determines the Gaussian from which the values are drawn for the observations that participate in the bicluster. "POS": The sign of the mean is fixed.

Sequentially Z_i are generated using `l`, `f1`, `of1`, `sd_z_noise`, `mean_z`, `sd_z`. `of1` gives the minimal samples participating in a bicluster to which between 0 and $l/f1$ samples are added, where the number is uniformly chosen. `sd_z_noise` gives the noise of samples not participating in the bicluster. `mean_z` and `sd_z` determines the Gaussian from which the values are drawn for the samples that participate in the bicluster.

U is the overall Gaussian zero mean noise generated by `sd_noise`.

Implementation in R.

Value

`X`	the noise data from $R^{n \times l}$.
`Y`	the noise free data from $R^{n \times l}$.
`ZC`	list where i-th element gives samples belonging to i-th bicluster.
`LC`	list where i-th element gives observations belonging to i-th bicluster.

Author(s)

Sepp Hochreiter

See Also

`fabia`, `fabias`, `fabiap`, `fabi`, `fabiasp`, `mfsc`, `nmfdiv`, `nmfeu`, `nmfsc`, `extractPlot`, `extractBic`, `plotBicluster`, `Factorization`, `projFuncPos`, `projFunc`, `estimateMode`, `makeFabiaData`, `makeFabiaDataBlocks`, `makeFabiaDataPos`, `makeFabiaDataBlocksPos`, `matrixImagePlot`, `fabiaDemo`, `fabiaVersion`

Examples

```
#---------------
# TEST
#---------------

dat <- makeFabiaDataPos(n = 100,l= 50,p = 3,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

X <- dat[[1]]
Y <- dat[[2]]

matrixImagePlot(Y)
dev.new()
matrixImagePlot(X)
```

```
## Not run:
#---------------
# DEMO
#---------------

dat <- makeFabiaDataPos(n = 1000,l= 100,p = 10,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

X <- dat[[1]]
Y <- dat[[2]]

matrixImagePlot(Y)
dev.new()
matrixImagePlot(X)


## End(Not run)
```

`matrixImagePlot` *Plotting of a Matrix*

Description

`matrixImagePlot`: R implementation of `myImagePlot`.

Usage

```
matrixImagePlot(x,xLabels=NULL, yLabels=NULL, zlim=NULL, title=NULL)
```

Arguments

`x`	the matrix.
`xLabels`	vector of strings to label the columns (default "colnames(x)").
`yLabels`	vector of strings to label the rows (default "rownames(x)").
`zlim`	vector containing a low and high value to use for the color scale.
`title`	title of the plot.

Details

Plotting a table of numbers as an image using R.

The color scale is based on the highest and lowest values in the matrix.

The original R code has been obtained by http://www.phaget4.org/R/myImagePlot.R and then has been modified.

Value

Plotting a table of numbers as an image

References

http://www.phaget4.org/R/myImagePlot.R

See Also

fabia, fabias, fabiap, fabi, fabiasp, mfsc, nmfdiv, nmfeu, nmfsc, extractPlot, extractBic, plotBicluster, Factorization, projFuncPos, projFunc, estimateMode, makeFabiaData, makeFabiaDataBlocks, makeFabiaDataPos, makeFabiaDataBlocksPos, matrixImagePlot, fabiaDemo, fabiaVersion

Examples

```
#---------------
# TEST
#---------------

dat <- makeFabiaDataBlocks(n = 100,l= 50,p = 3,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

X <- dat[[1]]
Y <- dat[[2]]

matrixImagePlot(Y)
dev.new()
matrixImagePlot(X)


## Not run:
#---------------
# DEMO
#---------------

dat <- makeFabiaDataBlocks(n = 1000,l= 100,p = 10,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

X <- dat[[1]]
Y <- dat[[2]]
X <- X- rowMeans(X)
XX <- (1/ncol(X))*tcrossprod(X)
dXX <- 1/sqrt(diag(XX)+0.001*as.vector(rep(1,nrow(X))))
X <- dXX*X

matrixImagePlot(X)


## End(Not run)
```

`mfsc` *Sparse Matrix Factorization for Bicluster Analysis (MFSC)*

Description

`mfsc`: R implementation of `mfsc`.

Usage

```
mfsc(X,p=5,cyc=100,sL=0.6,sZ=0.6,center=2,norm=1)
```

Arguments

`X`	the data matrix.
`p`	number of hidden factors = number of biclusters; default = 5.
`cyc`	maximal number of iterations; default = 100.
`sL`	final sparseness loadings; default = 0.6.
`sZ`	final sparseness factors; default = 0.6.
`center`	data centering: 1 (mean), 2 (median), > 2 (mode), 0 (no); default = 2.
`norm`	data normalization: 1 (0.75-0.25 quantile), >1 (var=1), 0 (no); default = 1.

Details

Biclusters are found by sparse matrix factorization where *both* factors are sparse.

Essentially the model is the sum of outer products of vectors:

$$X = \sum_{i=1}^{p} \lambda_i z_i^T$$

where the number of summands p is the number of biclusters. The matrix factorization is

$$X = LZ$$

Here λ_i are from R^n, z_i from R^l, L from $R^{n \times p}$, Z from $R^{p \times l}$, and X from $R^{n \times l}$.

No noise assumption: In contrast to factor analysis there is no noise assumption.

If the nonzero components of the sparse vectors are grouped together then the outer product results in a matrix with a nonzero block and zeros elsewhere.

The model selection is performed by a constraint optimization according to Hoyer, 2004. The Euclidean distance (the Frobenius norm) is minimized subject to sparseness constraints.

Model selection is done by gradient descent on the Euclidean objective and thereafter projection of single vectors of L and single vectors of Z to fulfill the sparseness constraints.

The projection minimize the Euclidean distance to the original vector given an l_1-norm and an l_2-norm.

The projection is a convex quadratic problem which is solved iteratively where at each iteration at least one component is set to zero. Instead of the l_1-norm a sparseness measurement is used which relates the l_1-norm to the l_2-norm.

The code is implemented in R.

Value

object of the class `Factorization`. Containing `LZ` (estimated noise free data LZ), `L` (loadings L), `Z` (factors Z), `U` (noise $X - LZ$), `center` (centering vector), `scaleData` (scaling vector), `X` (centered and scaled data X)

Author(s)

Sepp Hochreiter

References

S. Hochreiter et al., 'FABIA: Factor Analysis for Bicluster Acquisition', Bioinformatics 26(12):1520-1527, 2010. http://bioinformatics.oxfordjournals.org/cgi/content/abstract/btq227

Patrik O. Hoyer, 'Non-negative Matrix Factorization with Sparseness Constraints', Journal of Machine Learning Research 5:1457-1469, 2004.

See Also

fabia, fabias, fabiap, fabi, fabiasp, mfsc, nmfdiv, nmfeu, nmfsc, extractPlot, extractBic, plotBicluster, Factorization, projFuncPos, projFunc, estimateMode, makeFabiaData, makeFabiaDataBlocks, makeFabiaDataPos, makeFabiaDataBlocksPos, matrixImagePlot, fabiaDemo, fabiaVersion

Examples

```
#---------------
# TEST
#---------------

dat <- makeFabiaDataBlocks(n = 100,l= 50,p = 3,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

X <- dat[[1]]
Y <- dat[[2]]


resEx <- mfsc(X,3,30,0.6,0.6)

## Not run:

#-----------------
# DEMO1: Toy Data
```

```
#-----------------

n = 1000
l= 100
p = 10

dat <- makeFabiaDataBlocks(n = n,l= l,p = p,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

X <- dat[[1]]
Y <- dat[[2]]
ZC <- dat[[3]]
LC <- dat[[4]]

gclab <- rep.int(0,l)
gllab <- rep.int(0,n)
clab <- as.character(1:l)
llab <- as.character(1:n)
for (i in 1:p){
 for (j in ZC[i]){
     clab[j] <- paste(as.character(i),"_",clab[j],sep="")
 }
 for (j in LC[i]){
     llab[j] <- paste(as.character(i),"_",llab[j],sep="")
 }
 gclab[unlist(ZC[i])] <- gclab[unlist(ZC[i])] + p^i
 gllab[unlist(LC[i])] <- gllab[unlist(LC[i])] + p^i
}


groups <- gclab

#### MFSC

resToy4 <- mfsc(X,13,100,0.6,0.6)

extractPlot(resToy4,ti="MFSC",Y=Y)

raToy4 <- extractBic(resToy4,thresZ=0.01,thresL=0.05)

if ((raToy4$bic[[1]][1]>1) && (raToy4$bic[[1]][2])>1) {
    plotBicluster(raToy4,1)
}
if ((raToy4$bic[[2]][1]>1) && (raToy4$bic[[2]][2])>1) {
    plotBicluster(raToy4,2)
}
if ((raToy4$bic[[3]][1]>1) && (raToy4$bic[[3]][2])>1) {
    plotBicluster(raToy4,3)
}
if ((raToy4$bic[[4]][1]>1) && (raToy4$bic[[4]][2])>1) {
    plotBicluster(raToy4,4)
}
```

```
colnames(X(resToy4)) <- clab

rownames(X(resToy4)) <- llab


plot(resToy4,dim=c(1,2),label.tol=0.1,col.group = groups,lab.size=0.6)
plot(resToy4,dim=c(1,3),label.tol=0.1,col.group = groups,lab.size=0.6)
plot(resToy4,dim=c(2,3),label.tol=0.1,col.group = groups,lab.size=0.6)


#-----------------------------------------
# DEMO2: Laura van't Veer's gene expression
#        data set for breast cancer
#-----------------------------------------

avail <- require(fabiaData)

if (!avail) {
    message("")
    message("")
    message("########################################################")
    message("Package 'fabiaData' is not available: please install.")
    message("########################################################")
} else {


data(Breast_A)

X <- as.matrix(XBreast)

resBreast4 <- mfsc(X,5,100,0.6,0.6)

extractPlot(resBreast4,ti="MFSC Breast cancer(Veer)")


raBreast4 <- extractBic(resBreast4,thresZ=0.01,thresL=0.05)

if ((raBreast4$bic[[1]][1]>1) && (raBreast4$bic[[1]][2])>1) {
    plotBicluster(raBreast4,1)
}
if ((raBreast4$bic[[2]][1]>1) && (raBreast4$bic[[2]][2])>1) {
    plotBicluster(raBreast4,2)
}
if ((raBreast4$bic[[3]][1]>1) && (raBreast4$bic[[3]][2])>1) {
    plotBicluster(raBreast4,3)
}
if ((raBreast4$bic[[4]][1]>1) && (raBreast4$bic[[4]][2])>1) {
    plotBicluster(raBreast4,4)
}

plot(resBreast4,dim=c(1,2),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast4,dim=c(1,3),label.tol=0.03,col.group=CBreast,lab.size=0.6)
```

```
plot(resBreast4,dim=c(1,4),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast4,dim=c(1,5),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast4,dim=c(2,3),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast4,dim=c(2,4),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast4,dim=c(2,5),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast4,dim=c(3,4),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast4,dim=c(3,5),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast4,dim=c(4,5),label.tol=0.03,col.group=CBreast,lab.size=0.6)

}


#----------------------------------
# DEMO3: Su's multiple tissue types
#        gene expression data set
#----------------------------------


avail <- require(fabiaData)

if (!avail) {
    message("")
    message("")
    message("######################################################")
    message("Package 'fabiaData' is not available: please install.")
    message("######################################################")
} else {

data(Multi_A)

X <- as.matrix(XMulti)

resMulti4 <- mfsc(X,5,100,0.6,0.6)

extractPlot(resMulti4,ti="MFSC Multiple tissues(Su)")


raMulti4 <- extractBic(resMulti4,thresZ=0.01,thresL=0.05)

if ((raMulti4$bic[[1]][1]>1) && (raMulti4$bic[[1]][2])>1) {
    plotBicluster(raMulti4,1)
}
if ((raMulti4$bic[[2]][1]>1) && (raMulti4$bic[[2]][2])>1) {
    plotBicluster(raMulti4,2)
}
if ((raMulti4$bic[[3]][1]>1) && (raMulti4$bic[[3]][2])>1) {
    plotBicluster(raMulti4,3)
}
if ((raMulti4$bic[[4]][1]>1) && (raMulti4$bic[[4]][2])>1) {
    plotBicluster(raMulti4,4)
}

plot(resMulti4,dim=c(1,2),label.tol=0.01,col.group=CMulti,lab.size=0.6)
```

```
plot(resMulti4,dim=c(1,3),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti4,dim=c(1,4),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti4,dim=c(1,5),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti4,dim=c(2,3),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti4,dim=c(2,4),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti4,dim=c(2,5),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti4,dim=c(3,4),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti4,dim=c(3,5),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti4,dim=c(4,5),label.tol=0.01,col.group=CMulti,lab.size=0.6)

}



#----------------------------------------
# DEMO4: Rosenwald's diffuse large-B-cell
#        lymphoma gene expression data set
#----------------------------------------

avail <- require(fabiaData)

if (!avail) {
    message("")
    message("")
    message("#####################################################")
    message("Package 'fabiaData' is not available: please install.")
    message("#####################################################")
} else {


data(DLBCL_B)

X <- as.matrix(XDLBCL)


resDLBCL4 <- mfsc(X,5,100,0.6,0.6)

extractPlot(resDLBCL4,ti="MFSC Lymphoma(Rosenwald)")

raDLBCL4 <- extractBic(resDLBCL4,thresZ=0.01,thresL=0.05)

if ((raDLBCL4$bic[[1]][1]>1) && (raDLBCL4$bic[[1]][2])>1) {
    plotBicluster(raDLBCL4,1)
}
if ((raDLBCL4$bic[[2]][1]>1) && (raDLBCL4$bic[[2]][2])>1) {
    plotBicluster(raDLBCL4,2)
}
if ((raDLBCL4$bic[[3]][1]>1) && (raDLBCL4$bic[[3]][2])>1) {
    plotBicluster(raDLBCL4,3)
}
if ((raDLBCL4$bic[[4]][1]>1) && (raDLBCL4$bic[[4]][2])>1) {
    plotBicluster(raDLBCL4,4)
}
```

```
plot(resDLBCL4,dim=c(1,2),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL4,dim=c(1,3),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL4,dim=c(1,4),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL4,dim=c(1,5),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL4,dim=c(2,3),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL4,dim=c(2,4),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL4,dim=c(2,5),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL4,dim=c(3,4),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL4,dim=c(3,5),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL4,dim=c(4,5),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)

}



## End(Not run)
```

`nmfdiv` *Non-negative Matrix Factorization: Kullback-Leibler Divergence*

Description

`nmfdiv`: R implementation of `nmfdiv`.

Usage

```
nmfdiv(X,p=5,cyc=100)
```

Arguments

`X`	the data matrix.
`p`	number of hidden factors = number of biclusters; default = 5.
`cyc`	maximal number of iterations; default = 100.

Details

Non-negative Matrix Factorization represents positive matrix X by positive matrices L and Z.

Objective for reconstruction is Kullback-Leibler divergence.

Essentially the model is the sum of outer products of vectors:

$$X = \sum_{i=1}^{p} \lambda_i z_i^T$$

where the number of summands p is the number of biclusters. The matrix factorization is

$$X = LZ$$

Here λ_i are from R^n, z_i from R^l, L from $R^{n \times p}$, Z from $R^{p \times l}$, and X from $R^{n \times l}$.

The model selection is performed according to D. D. Lee and H. S. Seung, 1999, 2001.

The code is implemented in R.

Value

object of the class `Factorization`. Containing `LZ` (estimated noise free data LZ), `L` (loading L), `Z` (factors Z), `U` (noise $X - LZ$), `X` (scaled data X).

Author(s)

Sepp Hochreiter

References

D. D. Lee and H. S. Seung, 'Algorithms for non-negative matrix factorization', In Advances in Neural Information Processing Systems 13, 556-562, 2001.

D. D. Lee and H. S. Seung, 'Learning the parts of objects by non-negative matrix factorization', Nature, 401(6755):788-791, 1999.

See Also

fabia, fabias, fabiap, fabi, fabiasp, mfsc, nmfdiv, nmfeu, nmfsc, extractPlot, extractBic, plotBicluster, Factorization, projFuncPos, projFunc, estimateMode, makeFabiaData, makeFabiaDataBlocks, makeFabiaDataPos, makeFabiaDataBlocksPos, matrixImagePlot, fabiaDemo, fabiaVersion

Examples

```
#---------------
# TEST
#---------------

dat <- makeFabiaDataBlocks(n = 100,l= 50,p = 3,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

X <- dat[[1]]
Y <- dat[[2]]
X <- abs(X)


resEx <- nmfdiv(X,3)


## Not run:
```

```
#---------------
# DEMO
#---------------

dat <- makeFabiaDataBlocks(n = 1000,l= 100,p = 10,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

X <- dat[[1]]
Y <- dat[[2]]
X <- abs(X)


resToy <- nmfdiv(X,13)

extractPlot(resToy,ti="NMFDIV",Y=Y)


## End(Not run)
```

---

nmfeu *Non-negative Matrix Factorization: Euclidean Distance*

Description

nmfeu: R implementation of nmfeu.

Usage

```
nmfeu(X,p=5,cyc=100)
```

Arguments

X	the data matrix.
p	number of hidden factors = number of biclusters; default = 5.
cyc	maximal number of iterations; default = 100.

Details

Non-negative Matrix Factorization represents positive matrix X by positive matrices L and Z.

Objective for reconstruction is Euclidean distance.

Essentially the model is the sum of outer products of vectors:

$$X = \sum_{i=1}^{p} \lambda_i z_i^T$$

where the number of summands p is the number of biclusters. The matrix factorization is

$$X = LZ$$

Here λ_i are from R^n, z_i from R^l, L from $R^{n \times p}$, Z from $R^{p \times l}$, and X from $R^{n \times l}$.

The model selection is performed according to D. D. Lee and H. S. Seung, 2001.

The code is implemented in R.

Value

object of the class `Factorization`. Containing `LZ` (estimated noise free data LZ), `L` (loadings L), `Z` (factors Z), `U` (noise $X - LZ$), `X` (scaled data X).

Author(s)

Sepp Hochreiter

References

Paatero, P and Tapper, U, 'Least squares formulation of robust non-negative factor analysis', Chemometr. Intell. Lab. 37: 23-35, 1997.

D. D. Lee and H. S. Seung, 'Algorithms for non-negative matrix factorization', In Advances in Neural Information Processing Systems 13, 556-562, 2001.

See Also

fabia, fabias, fabiap, fabi, fabiasp, mfsc, nmfdiv, nmfeu, nmfsc, extractPlot, extractBic, plotBicluster, Factorization, projFuncPos, projFunc, estimateMode, makeFabiaData, makeFabiaDataBlocks, makeFabiaDataPos, makeFabiaDataBlocksPos, matrixImagePlot, fabiaDemo, fabiaVersion

Examples

```
#---------------
# TEST
#---------------

dat <- makeFabiaDataBlocks(n = 100,l= 50,p = 3,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

X <- dat[[1]]
Y <- dat[[2]]
X <- abs(X)


resEx <- nmfeu(X,3)


## Not run:
```

```
#---------------
# DEMO
#---------------

dat <- makeFabiaDataBlocks(n = 1000,l= 100,p = 10,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

X <- dat[[1]]
Y <- dat[[2]]
X <- abs(X)


resToy <- nmfeu(X,13)

extractPlot(resToy,ti="NMFEU",Y=Y)


## End(Not run)
```

`nmfsc` *Non-negative Sparse Matrix Factorization*

Description

`nmfsc`: R implementation of `nmfsc`.

Usage

```
nmfsc(X,p=5,cyc=100,sL=0.6,sZ=0.6)
```

Arguments

`X`	the data matrix.
`p`	number of hidden factors = number of biclusters; default = 5.
`cyc`	maximal number of iterations; default = 100.
`sL`	sparseness loadings; default = 0.6.
`sZ`	sparseness factors; default = 0.6.

Details

Non-negative Matrix Factorization represents positive matrix X by positive matrices L and Z that are sparse.

Objective for reconstruction is Euclidean distance and sparseness constraints.

Essentially the model is the sum of outer products of vectors:

$$X = \sum_{i=1}^{p} \lambda_i z_i^T$$

where the number of summands p is the number of biclusters. The matrix factorization is

$$X = LZ$$

Here λ_i are from R^n, z_i from R^l, L from $R^{n \times p}$, Z from $R^{p \times l}$, and X from $R^{n \times l}$.

If the nonzero components of the sparse vectors are grouped together then the outer product results in a matrix with a nonzero block and zeros elsewhere.

The model selection is performed by a constraint optimization according to Hoyer, 2004. The Euclidean distance (the Frobenius norm) is minimized subject to sparseness and non-negativity constraints.

Model selection is done by gradient descent on the Euclidean objective and thereafter projection of single vectors of L and single vectors of Z to fulfill the sparseness and non-negativity constraints.

The projection minimize the Euclidean distance to the original vector given an l_1-norm and an l_2-norm and enforcing non-negativity.

The projection is a convex quadratic problem which is solved iteratively where at each iteration at least one component is set to zero. Instead of the l_1-norm a sparseness measurement is used which relates the l_1-norm to the l_2-norm.

The code is implemented in R.

Value

object of the class `Factorization`. Containing `LZ` (estimated noise free data LZ), `L` (loadings L), `Z` (factors Z), `U` (noise $X - LZ$), `X` (data X).

Author(s)

Sepp Hochreiter

References

Patrik O. Hoyer, 'Non-negative Matrix Factorization with Sparseness Constraints', Journal of Machine Learning Research 5:1457-1469, 2004.

D. D. Lee and H. S. Seung, 'Algorithms for non-negative matrix factorization', In Advances in Neural Information Processing Systems 13, 556-562, 2001.

See Also

`fabia`, `fabias`, `fabiap`, `fabi`, `fabiasp`, `mfsc`, `nmfdiv`, `nmfeu`, `nmfsc`, `extractPlot`, `extractBic`, `plotBicluster`, `Factorization`, `projFuncPos`, `projFunc`, `estimateMode`, `makeFabiaData`, `makeFabiaDataBlocks`, `makeFabiaDataPos`, `makeFabiaDataBlocksPos`, `matrixImagePlot`, `fabiaDemo`, `fabiaVersion`

Examples

```
#---------------
# TEST
#---------------

dat <- makeFabiaDataBlocks(n = 100,l= 50,p = 3,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

X <- dat[[1]]
Y <- dat[[2]]
X <- abs(X)


resEx <- nmfsc(X,3,30,0.6,0.6)


## Not run:
#---------------
# DEMO
#---------------

dat <- makeFabiaDataBlocks(n = 1000,l= 100,p = 10,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

X <- dat[[1]]
Y <- dat[[2]]
X <- abs(X)


resToy <- nmfsc(X,13,100,0.6,0.6)

extractPlot(resToy,ti="NMFSC",Y=Y)


## End(Not run)
```

`plotBicluster` *Plotting of a bicluster*

Description

`plotBicluster`: R implementation of `plotBicluster`.

Usage

```
plotBicluster(r,p,opp=FALSE,zlim=NULL,title=NULL,which=c(1, 2))
```

Arguments

r	the result of extract_bic.
p	the bicluster to plot.
opp	plot opposite bicluster, default = FALSE.
zlim	vector containing a low and high value to use for the color scale.
title	title of the plot.
which	which plots are shown: 1=data matrix with bicluster on upper left, 2=plot of the bicluster; default c(1, 2).

Details

One bicluster is visualized by two plots. The variable "which" indicates which plots should be shown.

Plot1 (which=1): The data matrix is sorted such that the bicluster appear at the upper left corner. The bicluster is marked by a rectangle.

Plot2 (which=2): Only the bicluster is plotted.

Implementation in R.

Value

Plotting of a bicluster

Author(s)

Sepp Hochreiter

See Also

fabia, fabias, fabiap, fabi, fabiasp, mfsc, nmfdiv, nmfeu, nmfsc, extractPlot, extractBic, plotBicluster, Factorization, projFuncPos, projFunc, estimateMode, makeFabiaData, makeFabiaDataBlocks, makeFabiaDataPos, makeFabiaDataBlocksPos, matrixImagePlot, fabiaDemo, fabiaVersion

Examples

```
#---------------
# TEST
#---------------

dat <- makeFabiaDataBlocks(n = 100,l= 50,p = 3,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

X <- dat[[1]]
Y <- dat[[2]]


resEx <- fabia(X,3,0.01,20)
```

```
rEx <- extractBic(resEx)

plotBicluster(rEx,p=1)


## Not run:
#---------------
# DEMO1
#---------------

dat <- makeFabiaDataBlocks(n = 1000,l= 100,p = 10,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

X <- dat[[1]]
Y <- dat[[2]]


resToy <- fabia(X,13,0.01,200)



rToy <- extractBic(resToy)

plotBicluster(rToy,p=1)

#---------------
# DEMO2
#---------------

avail <- require(fabiaData)

if (!avail) {
    message("")
    message("")
    message("########################################################")
    message("Package 'fabiaData' is not available: please install.")
    message("########################################################")
} else {

data(Breast_A)

X <- as.matrix(XBreast)

resBreast <- fabia(X,5,0.1,200)

rBreast <- extractBic(resBreast)

plotBicluster(rBreast,p=1)

}
```

```
## End(Not run)
```

---

projFunc *Projection of a Vector to a Sparse Vector*

Description

projFunc: R implementation of projFunc.

Usage

```
projFunc(s, k1, k2)
```

Arguments

s	data vector.
k1	sparseness, l1 norm constraint.
k2	l2 norm constraint.

Details

The projection is done according to Hoyer, 2004: given an l_1-norm and an l_2-norm minimize the Euclidean distance to the original vector. The projection is a convex quadratic problem which is solved iteratively where at each iteration at least one component is set to zero.

In the applications, instead of the l_1-norm a sparseness measurement is used which relates the l_1-norm to the l_2-norm.

Implementation in R.

Value

v	sparse projected vector.

Author(s)

Sepp Hochreiter

References

Patrik O. Hoyer, 'Non-negative Matrix Factorization with Sparseness Constraints', Journal of Machine Learning Research 5:1457-1469, 2004.

See Also

fabia, fabias, fabiap, fabi, fabiasp, mfsc, nmfdiv, nmfeu, nmfsc, extractPlot, extractBic, plotBicluster, Factorization, projFuncPos, projFunc, estimateMode, makeFabiaData, makeFabiaDataBlocks, makeFabiaDataPos, makeFabiaDataBlocksPos, matrixImagePlot, fabiaDemo, fabiaVersion

Examples

```
#---------------
# DEMO
#---------------

size <- 30
sparseness <- 0.7

s <- as.vector(rnorm(size))
sp <- sqrt(1.0*size)-(sqrt(1.0*size)-1.0)*sparseness

ss <- projFunc(s,k1=sp,k2=1)

s
ss
```

---

projFuncPos *Projection of a Vector to a Non-negative Sparse Vector*

Description

projFuncPos: R implementation of projFuncPos.

Usage

```
projFuncPos(s, k1, k2)
```

Arguments

s	data vector.
k1	sparseness, l1 norm constraint.
k2	l2 norm constraint.

Details

The projection minimize the Euclidean distance to the original vector given an l_1-norm and an l_2-norm and enforcing non-negativity.

The projection is a convex quadratic problem which is solved iteratively where at each iteration at least one component is set to zero.

In the applications, instead of the l_1-norm a sparseness measurement is used which relates the l_1-norm to the l_2-norm:

Implementation in R.

Value

v non-negative sparse projected vector.

Author(s)

Sepp Hochreiter

References

Patrik O. Hoyer, 'Non-negative Matrix Factorization with Sparseness Constraints', Journal of Machine Learning Research 5:1457-1469, 2004.

See Also

fabia, fabias, fabiap, fabi, fabiasp, mfsc, nmfdiv, nmfeu, nmfsc, extractPlot, extractBic, plotBicluster, Factorization, projFuncPos, projFunc, estimateMode, makeFabiaData, makeFabiaDataBlocks, makeFabiaDataPos, makeFabiaDataBlocksPos, matrixImagePlot, fabiaDemo, fabiaVersion

Examples

```
#---------------
# DEMO
#---------------

size <- 30
sparseness <- 0.7

s <- as.vector(rnorm(size))
sp <- sqrt(1.0*size)-(sqrt(1.0*size)-1.0)*sparseness

ss <- projFuncPos(s,k1=sp,k2=1)

s
ss
```

---

readSamplesSpfabia *Factor Analysis for Bicluster Acquisition: Read Sparse Matrix Samples*

Description

readSamplesSpfabia: C implementation of readSamplesSpfabia.

Usage

```
readSamplesSpfabia(X,samples=0,lowerB=0.0,upperB=1000.0)
```

Arguments

X	the file name of the sparse matrix in sparse format.
samples	vector of samples which should be read; default = 0 (all samples)
lowerB	lower bound for filtering the inputs columns, the minimal column sum; default = 0.0.
upperB	upper bound for filtering the inputs columns, the maximal column sum; default = 1000.0.

Details

The data matrix is directly scanned by the C-code and must be in sparse matrix format.

Sparse matrix format: *first line: numer of rows (the samples). *second line: number of columns (the features). *following lines: for each sample (row) three lines with

I) number of nonzero row elements

II) indices of the nonzero row elements (ATTENTION: starts with 0!!)

III) values of the nonzero row elements (ATTENTION: floats with decimal point like 1.0 !!)

The code is implemented in C.

Value

X (data of the given samples)

Author(s)

Sepp Hochreiter

References

S. Hochreiter et al., 'FABIA: Factor Analysis for Bicluster Acquisition', Bioinformatics 26(12):1520-1527, 2010. http://bioinformatics.oxfordjournals.org/cgi/content/abstract/btq227

See Also

fabia, fabias, fabiap, spfabia, readSamplesSpfabia, readSpfabiaResult, fabi, fabiasp, mfsc, nmfdiv, nmfeu, nmfsc, extractPlot, extractBic, plotBicluster, Factorization, projFuncPos, projFunc, estimateMode, makeFabiaData, makeFabiaDataBlocks, makeFabiaDataPos, makeFabiaDataBlocksPos, matrixImagePlot, fabiaDemo, fabiaVersion

Examples

```
#---------------
# TEST
#---------------
```

readSpfabiaResult *Factor Analysis for Bicluster Acquisition: Read Results of SpFabia*

Description

readSpfabiaResult: C implementation of readSpfabiaResult.

Usage

```
readSpfabiaResult(X)
```

Arguments

X	the file prefix name of the result files of spfabia.

Details

Read the results of spfabia.

The code is implemented in C.

Value

object of the class Factorization. Containing L (loadings L), Z (factors Z), Psi (noise variance σ), lapla (variational parameter), avini (the information which the factor z_{ij} contains about x_j averaged over j) xavini (the information which the factor z_j contains about x_j) ini (for each j the information which the factor z_{ij} contains about x_j).

Author(s)

Sepp Hochreiter

References

S. Hochreiter et al., 'FABIA: Factor Analysis for Bicluster Acquisition', Bioinformatics 26(12):1520-1527, 2010. http://bioinformatics.oxfordjournals.org/cgi/content/abstract/btq227

See Also

fabia, fabias, fabiap, spfabia, readSamplesSpfabia, readSpfabiaResult, fabi, fabiasp, mfsc, nmfdiv, nmfeu, nmfsc, extractPlot, extractBic, plotBicluster, Factorization, projFuncPos, projFunc, estimateMode, makeFabiaData, makeFabiaDataBlocks, makeFabiaDataPos, makeFabiaDataBlocksPos, matrixImagePlot, fabiaDemo, fabiaVersion

samplesPerFeature *Factor Analysis for Bicluster Acquisition: Supplies samples per feature*

Description

samplesPerFeature: C implementation of samplesPerFeature.

Usage

```
samplesPerFeature(X,samples=0,lowerB=0.0,upperB=1000.0)
```

Arguments

X	the file name of the sparse matrix in sparse format.
samples	vector of samples which should be read; default = 0 (all samples)
lowerB	lower bound for filtering the inputs columns, the minimal column sum; default = 0.0.
upperB	upper bound for filtering the inputs columns, the maximal column sum; default = 1000.0.

Details

Supplies the samples for which a feature is not zero.

The data matrix is directly scanned by the C-code and must be in sparse matrix format.

Sparse matrix format: *first line: numer of rows (the samples). *second line: number of columns (the features). *following lines: for each sample (rows) three lines with

I) number of nonzero row elements

II) indices of the nonzero row elements (ATTENTION: starts with 0!!)

III) values of the nonzero row elements (ATTENTION: floats with decimal point like 1.0 !!)

The code is implemented in C.

Value

list with elements: sL (List with one element per feature: each element is a vector of samples where the feature is not zero.) nsL Vector of feature length containing number of samples having a non-zero feature value.

Author(s)

Sepp Hochreiter

References

S. Hochreiter et al., 'FABIA: Factor Analysis for Bicluster Acquisition', Bioinformatics 26(12):1520-1527, 2010. http://bioinformatics.oxfordjournals.org/cgi/content/abstract/btq227

See Also

fabia, fabias, fabiap, spfabia, readSamplesSpfabia, samplesPerFeature, readSpfabiaResult, fabi, fabiasp, mfsc, nmfdiv, nmfeu, nmfsc, extractPlot, extractBic, plotBicluster, Factorization, projFuncPos, projFunc, estimateMode, makeFabiaData, makeFabiaDataBlocks, makeFabiaDataPos, makeFabiaDataBlocksPos, matrixImagePlot, fabiaDemo, fabiaVersion

Examples

```
#---------------
# TEST
#---------------
```

---

spfabia *Factor Analysis for Bicluster Acquisition: SPARSE FABIA*

Description

spfabia: C implementation of spfabia.

Usage

```
spfabia(X,p=13,alpha=0.01,cyc=500,spl=0,spz=0.5,non_negative=0,random=1.0,write_file=1,norm=1,scale
```

Arguments

X	the file name of the sparse matrix in sparse format.
p	number of hidden factors = number of biclusters; default = 13.
alpha	sparseness loadings (0 - 1.0); default = 0.01.
cyc	number of iterations; default = 500.
spl	sparseness prior loadings (0 - 2.0); default = 0 (Laplace).
spz	sparseness factors (0.5 - 2.0); default = 0.5 (Laplace).
non_negative	Non-negative factors and loadings if non_negative > 0; default = 0.
random	>0: random initialization of loadings in [0,random], <0: random initialization of loadings in [-random,random]; default = 1.0.
write_file	>0: results are written to files (L in sparse format), default = 1.
norm	data normalization: >0 (var=1), 0 (no); default = 1.

scale	loading vectors are scaled in each iteration to the given variance. 0.0 indicates non scaling; default = 0.0.
lap	minimal value of the variational parameter; default = 1.0.
nL	maximal number of biclusters at which a row element can participate; default = 0 (no limit).
lL	maximal number of row elements per bicluster; default = 0 (no limit).
bL	cycle at which the nL or lL maximum starts; default = 0 (start at the beginning).
samples	vector of samples which should be included into the analysis; default = 0 (all samples)
initL	vector of indices of the selected samples which are used to initialize L; default = 0 (random initialization).
iter	number of iterations; default = 1.
quant	qunatile of largest L values to remove in each iteration; default = 0.001.
lowerB	lower bound for filtering the inputs columns, the minimal column sum; default = 0.0.
upperB	upper bound for filtering the inputs columns, the maximal column sum; default = 1000.0.
dorescale	rescale factors Z to variance 1 and consequently also L; logical; default: FALSE.
doini	compute the information content of the biclusters and sort the biclusters according to their information content; logical, default: FALSE.
eps	lower bound for variational parameter lapla; default: 1e-3.
eps1	lower bound for divisions to avoid division by zero; default: 1e-10.

Details

Version of fabia for a sparse data matrix. The data matrix is directly scanned by the C-code and must be in sparse matrix format.

Sparse matrix format: *first line: numer of rows (the samples). *second line: number of columns (the features). *following lines: for each sample (row) three lines with

I) number of nonzero row elements

II) indices of the nonzero row elements (ATTENTION: starts with 0!!)

III) values of the nonzero row elements (ATTENTION: floats with decimal point like 1.0 !!)

Biclusters are found by sparse factor analysis where *both* the factors and the loadings are sparse.

Essentially the model is the sum of outer products of vectors:

$$X = \sum_{i=1}^{p} \lambda_i z_i^T + U$$

where the number of summands p is the number of biclusters. The matrix factorization is

$$X = LZ + U$$

Here λ_i are from R^n, z_i from R^l, L from $R^{n \times p}$, Z from $R^{p \times l}$, and X, U from $R^{n \times l}$.

If the nonzero components of the sparse vectors are grouped together then the outer product results in a matrix with a nonzero block and zeros elsewhere.

The model selection is performed by a variational approach according to Girolami 2001 and Palmer et al. 2006.

We included a prior on the parameters and minimize a lower bound on the posterior of the parameters given the data. The update of the loadings includes an additive term which pushes the loadings toward zero (Gaussian prior leads to an multiplicative factor).

The code is implemented in C.

Value

object of the class `Factorization`. Containing `L` (loadings L), `Z` (factors Z), `Psi` (noise variance σ), `lapla` (variational parameter), `avini` (the information which the factor z_{ij} contains about x_j averaged over j) `xavini` (the information which the factor z_j contains about x_j) `ini` (for each j the information which the factor z_{ij} contains about x_j).

Author(s)

Sepp Hochreiter

References

S. Hochreiter et al., 'FABIA: Factor Analysis for Bicluster Acquisition', Bioinformatics 26(12):1520-1527, 2010. http://bioinformatics.oxfordjournals.org/cgi/content/abstract/btq227

Mark Girolami, 'A Variational Method for Learning Sparse and Overcomplete Representations', Neural Computation 13(11): 2517-2532, 2001.

J. Palmer, D. Wipf, K. Kreutz-Delgado, B. Rao, 'Variational EM algorithms for non-Gaussian latent variable models', Advances in Neural Information Processing Systems 18, pp. 1059-1066, 2006.

See Also

fabia, fabias, fabiap, spfabia, readSamplesSpfabia, samplesPerFeature, readSpfabiaResult, fabi, fabiasp, mfsc, nmfdiv, nmfeu, nmfsc, extractPlot, extractBic, plotBicluster, Factorization, projFuncPos, projFunc, estimateMode, makeFabiaData, makeFabiaDataBlocks, makeFabiaDataPos, makeFabiaDataBlocksPos, matrixImagePlot, fabiaDemo, fabiaVersion

Examples

```
#---------------
# TEST
#---------------

samples <- 20
features <- 200
sparseness <- 0.9
write(samples, file = "sparseFabiaTest.txt",ncolumns = features,append = FALSE, sep = " ")
write(features, file = "sparseFabiaTest.txt",ncolumns = features,append = TRUE, sep = " ")
for (i in 1:samples)
```

```
{
  ind <- which(runif(features)>sparseness)-1
  num <- length(ind)
  val <- abs(rnorm(num))
  write(num, file = "sparseFabiaTest.txt",ncolumns = features,append = TRUE, sep = " ")
  write(ind, file = "sparseFabiaTest.txt",ncolumns = features,append = TRUE, sep = " ")
  write(val, file = "sparseFabiaTest.txt",ncolumns = features,append = TRUE, sep = " ")
}

res <- spfabia("sparseFabiaTest",p=3,alpha=0.03,cyc=50,non_negative=1,write_file=0,norm=0)

unlink("sparseFabiaTest.txt")

plot(res,dim=c(1,2))
plot(res,dim=c(1,3))
plot(res,dim=c(2,3))
```

Index

∗ **EM algorithm**
fabi, 10
fabia, 13
fabiap, 21
fabias, 28
fabiasp, 34
readSamplesSpfabia, 76
readSpfabiaResult, 78
samplesPerFeature, 79
spfabia, 80
∗ **Laplace distribution**
fabi, 10
fabia, 13
fabiap, 21
fabias, 28
fabiasp, 34
readSamplesSpfabia, 76
readSpfabiaResult, 78
samplesPerFeature, 79
spfabia, 80
∗ **biclustering**
extractBic, 4
extractPlot, 7
fabi, 10
fabia, 13
fabiap, 21
fabias, 28
fabiasp, 34
Factorization-class, 39
makeFabiaData, 47
makeFabiaDataBlocks, 50
makeFabiaDataBlocksPos, 52
makeFabiaDataPos, 55
mfsc, 59
plotBicluster, 71
readSamplesSpfabia, 76
readSpfabiaResult, 78
samplesPerFeature, 79
spfabia, 80
∗ **classes**
Factorization-class, 39
∗ **cluster**
fabi, 10
fabia, 13
fabiap, 21
fabias, 28
fabiasp, 34
mfsc, 59
nmfsc, 69
readSamplesSpfabia, 76
readSpfabiaResult, 78
samplesPerFeature, 79
spfabia, 80
∗ **datagen**
makeFabiaData, 47
makeFabiaDataBlocks, 50
makeFabiaDataBlocksPos, 52
makeFabiaDataPos, 55
∗ **factor analysis**
fabi, 10
fabia, 13
fabiap, 21
fabias, 28
fabiasp, 34
readSamplesSpfabia, 76
readSpfabiaResult, 78
samplesPerFeature, 79
spfabia, 80
∗ **hplot**
extractPlot, 7
Factorization-class, 39
matrixImagePlot, 57
plotBicluster, 71
∗ **latent variables**
fabi, 10
fabia, 13
fabiap, 21
fabias, 28

fabiasp, 34
readSamplesSpfabia, 76
readSpfabiaResult, 78
samplesPerFeature, 79
spfabia, 80
∗ **manip**
Factorization-class, 39
∗ **matrix plot**
matrixImagePlot, 57
∗ **methods**
estimateMode, 2
fabi, 10
fabia, 13
fabiap, 21
fabias, 28
fabiasp, 34
Factorization-class, 39
mfsc, 59
nmfdiv, 65
nmfeu, 67
nmfsc, 69
projFunc, 74
projFuncPos, 75
readSamplesSpfabia, 76
readSpfabiaResult, 78
samplesPerFeature, 79
spfabia, 80
∗ **mode estimation**
estimateMode, 2
∗ **models**
fabiaVersion, 38
∗ **multivariate analysis**
fabi, 10
fabia, 13
fabiap, 21
fabias, 28
fabiasp, 34
readSamplesSpfabia, 76
readSpfabiaResult, 78
samplesPerFeature, 79
spfabia, 80
∗ **multivariate**
fabi, 10
fabia, 13
fabiap, 21
fabias, 28
fabiasp, 34
Factorization-class, 39
readSamplesSpfabia, 76
readSpfabiaResult, 78
samplesPerFeature, 79
spfabia, 80
∗ **non-negative matrix factorization**
extractBic, 4
extractPlot, 7
fabi, 10
fabia, 13
fabiap, 21
fabias, 28
mfsc, 59
nmfdiv, 65
nmfeu, 67
nmfsc, 69
projFuncPos, 75
readSamplesSpfabia, 76
readSpfabiaResult, 78
samplesPerFeature, 79
spfabia, 80
∗ **non-negative sparse coding**
projFuncPos, 75
∗ **nonnegative matrix factorization**
fabiasp, 34
∗ **sparse coding**
extractBic, 4
extractPlot, 7
fabi, 10
fabia, 13
fabiap, 21
fabias, 28
fabiasp, 34
makeFabiaData, 47
makeFabiaDataBlocks, 50
makeFabiaDataBlocksPos, 52
makeFabiaDataPos, 55
mfsc, 59
nmfdiv, 65
nmfeu, 67
nmfsc, 69
projFunc, 74
readSamplesSpfabia, 76
readSpfabiaResult, 78
samplesPerFeature, 79
spfabia, 80
∗ **sparse matrix factorization**
extractBic, 4
extractPlot, 7

makeFabiaData, 47
makeFabiaDataBlocks, 50
makeFabiaDataBlocksPos, 52
makeFabiaDataPos, 55
projFunc, 74

avini *(*Factorization-class*)*, 39
avini,Factorization-method *(*Factorization-class*)*, 39
avini<- *(*Factorization-class*)*, 39
avini<-,Factorization,numeric-method *(*Factorization-class*)*, 39
avini<-,Factorization,vector-method *(*Factorization-class*)*, 39

center *(*Factorization-class*)*, 39
center,Factorization-method *(*Factorization-class*)*, 39
center<- *(*Factorization-class*)*, 39
center<-,Factorization,numeric-method *(*Factorization-class*)*, 39
center<-,Factorization,vector-method *(*Factorization-class*)*, 39

estimateMode, 2, *3*, *5*, *8*, *11*, *15*, *20*, *23*, *29*, *36*, *39*, *45*, *49*, *51*, *54*, *56*, *58*, *60*, *66*, *68*, *70*, *72*, *74*, *76–78*, *80*, *82*
extractBic, *3*, 4, *5*, *8*, *11*, *15*, *20*, *23*, *29*, *36*, *39*, *45*, *49*, *51*, *54*, *56*, *58*, *60*, *66*, *68*, *70*, *72*, *74*, *76–78*, *80*, *82*
extractPlot, *3*, *5*, 7, *8*, *11*, *15*, *20*, *23*, *29*, *36*, *39*, *45*, *49*, *51*, *54*, *56*, *58*, *60*, *66*, *68*, *70*, *72*, *74*, *76–78*, *80*, *82*

fabi, *3*, *5*, *8*, 10, *11*, *15*, *20*, *23*, *29*, *36*, *39*, *45*, *49*, *51*, *54*, *56*, *58*, *60*, *66*, *68*, *70*, *72*, *74*, *76–78*, *80*, *82*
fabia, *3*, *5*, *8*, *11*, 13, *15*, *20*, *23*, *29*, *36*, *39*, *45*, *49*, *51*, *54*, *56*, *58*, *60*, *66*, *68*, *70*, *72*, *74*, *76–78*, *80*, *82*
fabiaDemo, *3*, *5*, *8*, *11*, *15*, *20*, 20, *23*, *29*, *36*, *39*, *45*, *49*, *51*, *54*, *56*, *58*, *60*, *66*, *68*, *70*, *72*, *74*, *76–78*, *80*, *82*
fabiap, *3*, *5*, *8*, *11*, *15*, *20*, 21, *23*, *29*, *36*, *39*, *45*, *49*, *51*, *54*, *56*, *58*, *60*, *66*, *68*, *70*, *72*, *74*, *76–78*, *80*, *82*
fabias, *3*, *5*, *8*, *11*, *15*, *20*, *23*, 27, *29*, *36*, *39*, *45*, *49*, *51*, *54*, *56*, *58*, *60*, *66*, *68*, *70*, *72*, *74*, *76–78*, *80*, *82*
fabiasp, *3*, *5*, *8*, *11*, *15*, *20*, *23*, *29*, 34, *36*, *39*, *45*, *49*, *51*, *54*, *56*, *58*, *60*, *66*, *68*, *70*, *72*, *74*, *76–78*, *80*, *82*
fabiaVersion, *3*, *5*, *8*, *11*, *15*, *20*, *23*, *29*, *36*, 38, *39*, *45*, *49*, *51*, *54*, *56*, *58*, *60*, *66*, *68*, *70*, *72*, *74*, *76–78*, *80*, *82*
Factorization, *3*, *5*, *8*, *11*, *15*, *20*, *23*, *29*, *36*, *39*, *45*, *49*, *51*, *54*, *56*, *58*, *60*, *66*, *68*, *70*, *72*, *74*, *76–78*, *80*, *82*
Factorization *(*Factorization-class*)*, 39
Factorization,ANY-method *(*Factorization-class*)*, 39
Factorization,list-method,numeric-method,vector-method,mat *(*Factorization-class*)*, 39
Factorization-class, 39
Factorization-method *(*Factorization-class*)*, 39

ini *(*Factorization-class*)*, 39
ini,Factorization-method *(*Factorization-class*)*, 39
ini<- *(*Factorization-class*)*, 39
ini<-,Factorization,matrix-method *(*Factorization-class*)*, 39

L *(*Factorization-class*)*, 39
l *(*Factorization-class*)*, 39
L,Factorization-method *(*Factorization-class*)*, 39
l,Factorization-method *(*Factorization-class*)*, 39
L<- *(*Factorization-class*)*, 39
l<- *(*Factorization-class*)*, 39
L<-,Factorization,matrix-method *(*Factorization-class*)*, 39
l<-,Factorization,numeric-method *(*Factorization-class*)*, 39
lapla *(*Factorization-class*)*, 39
lapla,Factorization-method *(*Factorization-class*)*, 39
lapla<- *(*Factorization-class*)*, 39
lapla<-,Factorization,matrix-method *(*Factorization-class*)*, 39
LZ *(*Factorization-class*)*, 39
LZ,Factorization-method *(*Factorization-class*)*, 39
LZ<- *(*Factorization-class*)*, 39
LZ<-,Factorization,matrix-method *(*Factorization-class*)*, 39

M (Factorization-class), 39
M,Factorization-method
(Factorization-class), 39
M<- (Factorization-class), 39
M<-,Factorization,matrix-method
(Factorization-class), 39
makeFabiaData, 3, 5, 8, 11, 15, 20, 23, 29, 36, 39, 45, 47, 49, 51, 54, 56, 58, 60, 66, 68, 70, 72, 74, 76–78, 80, 82
makeFabiaDataBlocks, 3, 5, 8, 11, 15, 20, 23, 29, 36, 39, 45, 49, 50, 51, 54, 56, 58, 60, 66, 68, 70, 72, 74, 76–78, 80, 82
makeFabiaDataBlocksPos, 3, 5, 8, 11, 15, 20, 23, 29, 36, 39, 45, 49, 51, 52, 54, 56, 58, 60, 66, 68, 70, 72, 74, 76–78, 80, 82
makeFabiaDataPos, 3, 5, 8, 11, 15, 20, 23, 29, 36, 39, 45, 49, 51, 54, 55, 56, 58, 60, 66, 68, 70, 72, 74, 76–78, 80, 82
matrixImagePlot, 3, 5, 8, 11, 15, 20, 23, 29, 36, 39, 45, 49, 51, 54, 56, 57, 58, 60, 66, 68, 70, 72, 74, 76–78, 80, 82
mfsc, 3, 5, 8, 11, 15, 20, 23, 29, 36, 39, 45, 49, 51, 54, 56, 58, 59, 60, 66, 68, 70, 72, 74, 76–78, 80, 82

n (Factorization-class), 39
n,Factorization-method
(Factorization-class), 39
n<- (Factorization-class), 39
n<-,Factorization,numeric-method
(Factorization-class), 39
nmfdiv, 3, 5, 8, 11, 15, 20, 23, 29, 36, 39, 45, 49, 51, 54, 56, 58, 60, 65, 66, 68, 70, 72, 74, 76–78, 80, 82
nmfeu, 3, 5, 8, 11, 15, 20, 23, 29, 36, 39, 45, 49, 51, 54, 56, 58, 60, 66, 67, 68, 70, 72, 74, 76–78, 80, 82
nmfsc, 3, 5, 8, 11, 15, 20, 23, 29, 36, 39, 45, 49, 51, 54, 56, 58, 60, 66, 68, 69, 70, 72, 74, 76–78, 80, 82

p1 (Factorization-class), 39
p1,Factorization-method
(Factorization-class), 39
p1<- (Factorization-class), 39
p1<-,Factorization,numeric-method
(Factorization-class), 39
p2 (Factorization-class), 39
p2,Factorization-method
(Factorization-class), 39
p2<- (Factorization-class), 39
p2<-,Factorization,numeric-method
(Factorization-class), 39
parameters (Factorization-class), 39
parameters,Factorization-method
(Factorization-class), 39
parameters<- (Factorization-class), 39
parameters<-,Factorization,list-method
(Factorization-class), 39
plot,Factorization,missing-method
(Factorization-class), 39
plot,Factorization-method
(Factorization-class), 39
plotBicluster, 3, 5, 8, 11, 15, 20, 23, 29, 36, 39, 45, 49, 51, 54, 56, 58, 60, 66, 68, 70, 71, 72, 74, 76–78, 80, 82
projFunc, 3, 5, 8, 11, 15, 20, 23, 29, 36, 39, 45, 49, 51, 54, 56, 58, 60, 66, 68, 70, 72, 74, 74, 76–78, 80, 82
projFuncPos, 3, 5, 8, 11, 15, 20, 23, 29, 36, 39, 45, 49, 51, 54, 56, 58, 60, 66, 68, 70, 72, 74, 75, 76–78, 80, 82
Psi (Factorization-class), 39
Psi,Factorization-method
(Factorization-class), 39
Psi<- (Factorization-class), 39
Psi<-,Factorization,numeric-method
(Factorization-class), 39
Psi<-,Factorization,vector-method
(Factorization-class), 39

readSamplesSpfabia, 76, 77, 78, 80, 82
readSpfabiaResult, 15, 39, 77, 78, 78, 80, 82

samplesPerFeature, 79, 80, 82
scaleData (Factorization-class), 39
scaleData,Factorization-method
(Factorization-class), 39
scaleData<- (Factorization-class), 39
scaleData<-,Factorization,numeric-method
(Factorization-class), 39
scaleData<-,Factorization,vector-method
(Factorization-class), 39
show,Factorization-method
(Factorization-class), 39
showSelected (Factorization-class), 39

showSelected,Factorization,numeric-method
(Factorization-class), 39
showSelected,Factorization-method
(Factorization-class), 39
spfabia, 8, 11, 15, 23, 29, 36, 39, 77, 78, 80, 80, 82
summary,Factorization-method
(Factorization-class), 39

U (Factorization-class), 39
U,Factorization-method
(Factorization-class), 39
U<- (Factorization-class), 39
U<-,Factorization,matrix-method
(Factorization-class), 39

X (Factorization-class), 39
X,Factorization-method
(Factorization-class), 39
X<- (Factorization-class), 39
X<-,Factorization,matrix-method
(Factorization-class), 39
xavini (Factorization-class), 39
xavini,Factorization-method
(Factorization-class), 39
xavini<- (Factorization-class), 39
xavini<-,Factorization,numeric-method
(Factorization-class), 39
xavini<-,Factorization,vector-method
(Factorization-class), 39

Z (Factorization-class), 39
Z,Factorization-method
(Factorization-class), 39
Z<- (Factorization-class), 39
Z<-,Factorization,matrix-method
(Factorization-class), 39