Package 'scBFA'

April 18, 2024

Version 1.17.0

Date 2019-03-09

Title A dimensionality reduction tool using gene detection pattern to mitigate noisy expression profile of scRNA-seq

Description This package is designed to model gene detection pattern of scRNA-seq through a binary factor analysis model. This model allows user to pass into a cell level covariate matrix X and gene level covariate matrix Q to account for nuisance variance(e.g batch effect), and it will output a low dimensional embedding matrix for downstream analysis.

URL https://github.com/ucdavis/quon-titative-biology/BFA

BugReports https://github.com/ucdavis/quon-titative-biology/BFA/issues

biocViews SingleCell, Transcriptomics, DimensionReduction,GeneExpression, ATACSeq, BatchEffect, KEGG, QualityControl

Depends R (>= 3.6)

Imports SingleCellExperiment, SummarizedExperiment, Seurat, MASS, zinbwave, stats, copula, ggplot2, DESeq2, utils, grid, methods, Matrix

Suggests knitr, rmarkdown, testthat, Rtsne

VignetteBuilder knitr

RoxygenNote 7.0.2

License GPL-3 + file LICENSE

LazyData true

Encoding UTF-8

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

git_branch devel

git_last_commit 142f0b0

git_last_commit_date 2023-10-24

Repository Bioconductor 3.19

Date/Publication 2024-04-17

Author Ruoxin Li [aut, cre],
Gerald Quon [aut]

Maintainer Ruoxin Li <uskli@ucdavis.edu>

Contents

BinaryPCA 2
celltype 4
celltype_toy 4
diagnose 5
disperPlot 6
exprdata 6
getGeneExpr 7
getLoading 7
getScore 8
gradient 8
gradient_chunk 9
InitBinaryFA 9
neg_loglikelihood 10
neg_loglikelihood_chunk 11
OptimBFA 11
restore 12
scBFA 13
scNoiseSim 15
zinb_toy 16

Index **17**

`BinaryPCA` *Performs Binary PCA (as outlined in our paper). This function take the input of gene expression profile and perform PCA on gene detection pattern*

Description

Performs Binary PCA (as outlined in our paper). This function take the input of gene expression profile and perform PCA on gene detection pattern

Usage

```
BinaryPCA(scData, X = NULL, scale. = FALSE, center = TRUE)
```

Arguments

scData	can be a raw count matrix, in which rows are genes and columns are cells; can be a seurat object; can be a SingleCellExperiment object.
X	N by C covariate matrix,e.g batch effect, in which rows are cells,columns are number of covariates. If no such covariates available X = NULL
scale.	Logical value isndicating whether the variables should be scaled to have unit variance before the analysis takes place. In general scaling is not advisable, since we think the variance in the gene detection space is potentially associated with celltypes (e.g cell type specific markers)
center	Logical value indicating whether the variables should be shifted to be zero centered

Value

A list with class "prcomp",containing the following components:

sdev: the standard deviations of the principal components (i.e., the square roots of the eigenvalues of the covariance/correlation matrix, though the calculation is actually done with the singular values of the data matrix).

rotation: the matrix of variable loadings (i.e., a matrix whose columns contain the eigenvectors). The function princomp returns this in the element loadings.

x: the rotated data (the centred (and scaled if requested) data multiplied by the rotation matrix) is returned. Hence, cov(x) is the diagonal matrix diag(sdev^2).

center, scale. centering and scaling used, or FALSE.

Examples

```
## Working with Seurat or SingleCellExperiment object

library(Seurat)
library(SingleCellExperiment)

## Input expression profile, 5 genes x 3 cells

GeneExpr = matrix(rpois(15,1),nrow = 5,ncol = 3)
rownames(GeneExpr) = paste0("gene",seq_len(nrow(GeneExpr)))
colnames(GeneExpr) = paste0("cell",seq_len(ncol(GeneExpr)))
celltype = as.factor(sample(c(1,2,3),3,replace = TRUE))

## Create cell level technical batches

batch = sample(c("replicate 1","replicate 2","replicate 2"))
X = matrix(NA,nrow = length(batch),ncol = 1)
X[which(batch =="replicate 1"), ] = 0
X[which(batch =="replicate 2"), ] = 1
rownames(X) = colnames(GeneExpr)

##run BFA with raw count matrix
```

```
bpca_model = BinaryPCA(scData = GeneExpr,X = scale(X))

## Create Seurat object for input to BFA

scData = CreateSeuratObject(counts = GeneExpr,project = "sc",min.cells = 0)

## Standardize the covariate matrix should be a default operation

bpca_model = BinaryPCA(scData = scData, X = scale(X))

## Build the SingleCellExperiment object for input to BFA

## Set up SingleCellExperiment class

sce <- SingleCellExperiment(assay = list(counts = GeneExpr))

## Standardize the covariate matrix should be a default operation

bpca_model = BinaryPCA(scData = sce, X = scale(X))
```

`celltype` *Cell types as labels of example scRNA-seq dataset(exprdata)*

Description

A vector contains the cell types as labels for cells in example scRNA-seq dataset(exprdata)

Usage

```
data(celltype)
```

References

https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE89232

`celltype_toy` *toy cell type vector with 3 cell types generated for 5 cells in toy dataset*

Description

The cell type vector is generated from the following code

Usage

```
data(celltype_toy)
```

Details

celltype = as.factor(sample(c(1,2,3),5,replace = TRUE))

diagnose *Perform diagnoisis of dispersion on the expression profile to check whether scBFA works on specific dataset*

Description

Perform diagnoisis of dispersion on the expression profile to check whether scBFA works on specific dataset

Usage

```
diagnose(
  scData,
  sampleInfo = NULL,
  disperType = "Fitted",
  diagnose_feature = "dispersion"
)
```

Arguments

scData	can be a raw count matrix, in which rows are genes and columns are cells; can be a seurat object; can be a SingleCellExperiment object.
sampleInfo	sample level feature matrix,e.g batch effect,experimental conditions in which rows are cells,columns are number of covariates.Default is NULL
disperType	a parameter to tell which dispersion estimate the user can plot DESeq2 offers stepwise dispersion estimate, a gene wise dispersion estimate using "GeneEst", dispersion estimate from fitted disperions ~ mean curve (using "Fitted") And final MAP estimate,using "Map". Default value is "Fitted"
diagnose_feature	a parameter to determine whether the user want to check GDR or dispersion.

Value

A Figure to tell the where the input data's dispersion ~ tpm curve align to the 14 benchmark datasets in Figure 2.a or Gene detection rate

Examples

```
data(exprdata)
diagnose(scData = exprdata)
```

disperPlot *Reference dataset(disperPlot)*

Description

A dataframe contains all the gene-wise dispersion estimates loess curve for 14 datasets we benchmarked in Figure 2.a

Usage

```
data(disperPlot)
```

Details

The variable in the columns are: fitted_dispersion: the log value of gene-wise dispersion after fitting a loess curve with respect to TPM value. Note that the genes at the top 2.5 meantpm is average tpm value calculated per gene dataset are nams for datasets variance is gene selection method, here is HEG vs HVG

exprdata *scRNA-seq dataset(exprdata)*

Description

A matrix contains 950 cells and 500 genes. The source of this dataset is cDC/ pre-DC cells(see supplementary files) We subset most variant 100 genes as example scRNA-seq dataset(exprdata)

Usage

```
data(exprdata)
```

References

https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE89232

getGeneExpr *Function to extract gene expression matrix from input observation matrix*

Description

Function to extract gene expression matrix from input observation matrix

Usage

```
getGeneExpr(scData)
```

Arguments

scData	can be a raw count matrix, in which rows are genes and columns are cells; can be a seurat object; can be a SingleCellExperiment object

Value

a raw expression matrix in which rows are genes and columns are cells.

Examples

```
scData = matrix(rpois(15,1),3,5)

GeneExpr = getGeneExpr(scData)
```

getLoading *Function to get low dimensional loading matrix*

Description

Function to get low dimensional loading matrix

Usage

```
getLoading(modelEnv)
```

Arguments

modelEnv	output environment variable

Value

A: G by K compressed feature space

Examples

```
GeneExpr = matrix(rpois(15,1),3,5)
bfa_model = scBFA(scData = GeneExpr,X = NULL,numFactors =2)
A = getLoading(bfa_model)
```

`getScore` *Function to get low dimensional embedding matrix*

Description

Function to get low dimensional embedding matrix

Usage

```
getScore(modelEnv)
```

Arguments

`modelEnv`	output environment variable

Value

Z: N by K low dimensional embedding

Examples

```
GeneExpr = matrix(rpois(15,1),3,5)
bfa_model = scBFA(scData = GeneExpr,X = NULL,numFactors =2)
Z = getScore(bfa_model)
```

`gradient` *Calculate gradient of the negative log likelihood, used for calls to the optim() function.*

Description

Calculate gradient of the negative log likelihood, used for calls to the optim() function.

Usage

```
gradient(parameters, modelEnv)
```

Arguments

`parameters`	Vectorized parameter space.
`modelEnv`	Environment variable contains parameter space, and global variables such as N,G,C,detection matrix B, etc

Value

Vectorized gradient

`gradient_chunk` *Calculate gradient of the negative log likelihood, used for calls to the optim() function.*

Description

Calculate gradient of the negative log likelihood, used for calls to the optim() function.

Usage

```
gradient_chunk(parameters, modelEnv)
```

Arguments

`parameters`	Vectorized parameter space.
`modelEnv`	Environment variable contains parameter space, and global variables such as N,G,C,detection matrix B, etc

Value

Vectorized gradient

`InitBinaryFA` *This function should be called to initialize input parameters into the main scBFA function*

Description

This function should be called to initialize input parameters into the main scBFA function

Usage

```
InitBinaryFA(
  modelEnv,
  GeneExpr,
  numFactors,
  epsilon,
  X = NULL,
  Q = NULL,
  initCellcoef,
  updateCellcoef,
  updateGenecoef,
  NUM_CELLS_PER_CHUNK = min(ncol(GeneExpr), 50000),
  doChunking = (NUM_CELLS_PER_CHUNK < modelEnv$numCells)
)
```

Arguments

modelEnv	Empty R environment variable to contain following parameters: A,Z,V,U,β,γ,ϵ
GeneExpr	G by N rawcount matrix, in which rows are genes and columns are cells
numFactors	Numeric value, number of latent dimensions
epsilon	Numeric value, parameter to control the strength of regularization
X	N by C cell-specific covariate matrix(e.g batch effect), in which rows are cells,columns are number of covariates. If no such covariates are available, then X = NULL
Q	G by T gene-specific covariate matrix(e.g quality control measures), in which rows are genes columns are number of covariates, If no such covariates are available, then Q = NULL
initCellcoef	Initialization of C by G gene-specific coefficient matrix as user-defined coefficient β. Such user defined coefficient can be applied to address confounding batch effect
updateCellcoef	Logical value, parameter to decide whether to update C by G gene-specific coefficient matrix. Again, when the cell types are confounded with technical batches or there is no cell level covariate matrix, the user can keep the initialization of coefficients as known estimate.
updateGenecoef	Logical value, parameter to decide whether to update N by T gene-specific coefficient matrix. Again, when there is no gene level covariate matrix, this value should be FALSE by default.
NUM_CELLS_PER_CHUNK	scBFA can run out of memory on large datasets, so we can chunk up computations to avoid this if necessary. NUM_CELLS_PER_CHUNK is the number of cells per 'chunk' computed. Shrink if running out of mem.
doChunking	Use memory-efficient (but slower) chunking. Will do automatically if the chunk size is specified to be smaller than the # of cells in dataset.

Value

A model environment containing the following parameters: A,Z,V,U,β,γ,ϵ.

---

neg_loglikelihood *Calculate negative penalized likelihood, used for calls to the optim() function.*

Description

The penalized likelihood function: $f(A, Z, \beta, O, U) = \sum[lnP(B; A, Z, U, V, \beta, \gamma)]_i j - \epsilon_1 * ||A||_2^2 - \epsilon_2 * ||Z||_2^2 - \epsilon_3 * ||\beta||_2^2 - \epsilon_2 * ||\gamma||_2^2$

Usage

```
neg_loglikelihood(parameters, modelEnv)
```

Arguments

parameters	Vectorized parameter space.
modelEnv	Environment variable contains parameter space, and global variables such as N,G,C,detection matrix B, etc

Value

Scalar penalized likelihood

---

neg_loglikelihood_chunk

Calculate negative penalized likelihood, used for calls to the optim() function.

Description

The penalized likelihood function: $f(A, Z, \beta, O, U) = \sum[lnP(B; A, Z, U, V, \beta, \gamma)]_i j - \epsilon_1 * ||A||_2^2 - \epsilon_2 * ||Z||_2^2 - \epsilon_3 * ||\beta||_2^2 - \epsilon_2 * ||\gamma||_2^2$

Usage

```
neg_loglikelihood_chunk(parameters, modelEnv)
```

Arguments

parameters	Vectorized parameter space.
modelEnv	Environment variable contains parameter space, and global variables such as N,G,C,detection matrix B, etc

Value

Scalar penalized likelihood

OptimBFA

Optimize parameters of BFA's likelihood function

Description

Optimize parameters of BFA's likelihood function

Usage

```
OptimBFA(modelEnv, maxit, method)
```

Arguments

modelEnv	Environment variable contains parameter space, and global variables such as N,G,C,detection matrix B, etc
maxit	Maximum number of iteration with respect to objective function, default is 300 iterations
method	Optimization method, default is the conjugate gradient approach L-BFGS-B is recommended for smaller dataset less than 10k cells

Value

The entire model environment

---

restore *Restore the vector of parameter space into their seperated parameterization*

Description

Restore the vector of parameter space into their seperated parameterization

Usage

```
restore(parameters, modelEnv)
```

Arguments

parameters:	Vectorized parameter space.
modelEnv:	Environment variable contains parameter space , and global variables such as N,G,C,T,detection matrix B etc

Value

A list parameters containing the following parameters: A,Z,U,V,beta,gamma,epsilon

scBFA *Perform BFA model on the expression profile*

Description

Perform BFA model on the expression profile

Usage

```
scBFA(
  scData,
  numFactors,
  X = NULL,
  Q = NULL,
  maxit = 300,
  method = "L-BFGS-B",
  initCellcoef = NULL,
  updateCellcoef = TRUE,
  updateGenecoef = TRUE,
  NUM_CELLS_PER_CHUNK = 5000,
  doChunking = FALSE
)
```

Arguments

scData	can be a raw count matrix, in which rows are genes and columns are cells; can be a seurat object; can be a SingleCellExperiment object.
numFactors	Numeric value, number of latent dimensions
X	N by C covariate matrix,e.g batch effect, in which rows are cells,columns are number of covariates.Default is NULL
Q	G by T gene-specific covariate matrix(e.g quality control measures), in which rows are genes columns are number of covariates, If no such covariates are available, then Q = NULL
maxit	Numeric value, parameter to control the Maximum number of iterations in the optimization, default is 300.
method	Method of optimization,default is L-BFGS-B(Limited memory BFGS) approach. Conjugate Gradient (CG) is recommended for larger dataset (number of cells > 10k)
initCellcoef	Initialization of C by G gene-specific coefficient matrix as user-defined coefficient β. Such user defined coefficient can be applied to address confounding batch effect
updateCellcoef	Logical value, parameter to decide whether to update C by G gene-specific coefficient matrix. Again, when the cell types are confounded with technical batches or there is no cell level covariate matrix, the user can keep the initialization of coefficients as known estimate.

updateGenecoef Logical value, parameter to decide whether to update N by T gene-specific coefficient matrix. Again, when there is no gene level covariate matrix, this value should be FALSE by default.

NUM_CELLS_PER_CHUNK
scBFA can run out of memory on large datasets, so we can chunk up computations to avoid this if necessary. NUM_CELLS_PER_CHUNK is the number of cells per 'chunk' computed. Shrink if running out of mem.

doChunking Use memory-efficient (but slower) chunking. Will do automatically if the chunk size is specified to be smaller than the # of cells in dataset.

Value

A model environment containing all parameter space of a BFA model as well as global variables needed for calculation:

A: G by K compressed feature space matrix

Z: N by K low dimensional embedding matrix

β: C by G cell level coefficient matrix

γ: N by T gene level coefficient matrix

V: G by 1 offset matrix

U: N by 1 offset matrix

Examples

```
## Working with Seurat or SingleCellExperiment object

library(Seurat)
library(SingleCellExperiment)


## Input expression profile, 5 genes x 3 cells

GeneExpr = matrix(rpois(15,1),nrow = 5,ncol = 3)
rownames(GeneExpr) = paste0("gene",seq_len(nrow(GeneExpr)))
colnames(GeneExpr) = paste0("cell",seq_len(ncol(GeneExpr)))
celltype = as.factor(sample(c(1,2,3),3,replace = TRUE))

## Create cell level technical batches

batch = sample(c("replicate 1","replicate 2","replicate 2"))
X = matrix(NA,nrow = length(batch),ncol = 1)
X[which(batch =="replicate 1"), ] = 0
X[which(batch =="replicate 2"), ] = 1
rownames(X) = colnames(GeneExpr)

## run BFA with raw count matrix

bfa_model = scBFA(scData = GeneExpr,X = scale(X),numFactors =2)

## Create Seurat object for input to BFA
```

```
scData = CreateSeuratObject(counts = GeneExpr,project="sc",min.cells = 0)

## Standardize the covariate matrix should be a default operation

bfa_model = scBFA(scData = scData, X = scale(X), numFactors = 2)

## Build the SingleCellExperiment object for input to BFA

## Set up SingleCellExperiment class

sce <- SingleCellExperiment(assay = list(counts = GeneExpr))

## Standardize the covariate matrix should be a default operation

bfa_model = scBFA(scData = sce, X = scale(X), numFactors = 2)
```

`scNoiseSim` *simulation to generate scRNA-seq data with varying level of gene detection noise versus gene count noise*

Description

simulation to generate scRNA-seq data with varying level of gene detection noise versus gene count noise

Usage

```
scNoiseSim(zinb, celltype, disper, var_dropout = 1, var_count = 1, delta)
```

Arguments

`zinb`	a ZINB-WaVE object representing ZINB-WaVE fit to real data to get realistic simulation parameters
`celltype`	a factor to specify the ground-truth cell types in the original dataset that the parameter of zinb object is fit to. Since we filter out some simulated cells due to low amount of genes detected in that cell, we subset the ground truth cell types correspondingly
`disper`	numeric value, parameter to control the size factor r in $NB(\mu, r)$. r is varied in the set 0.5,1,5 in our simulation(as outlined in our paper)
`var_dropout`	numeric value, parameter to control the noise level added to a common embedding space for to generate gene detection matrix. This parameter is formulated as σ_π and in the paper is selected from the set 0.1, 0.5, 1, 2, 3
`var_count`	numeric value, parameter to control the noise level added to a common embedding space to generate gene count matrix. This parameter is formulated as σ_μ and and in the paper is selected from the set 0.1, 0.5, 1, 2, 3
`delta`	intercept to control the overall gene detection rate. and in the paper is selected from the set -2, -0.5, 1,2.5,4

Value

GeneExpr,a count matrix with rows number of genes and columns number of cells

celltype,a vector specify the corresponding celltype after QC measures.

Examples

```
## raw counts matrix with rows are genes and columns are cells
data("zinb_toy",package = "scBFA", envir = environment())
## a vector specify the ground truth of cell types provided by conquer database
data("celltype_toy",package = "scBFA",envir = environment())

scData = scNoiseSim(zinb = zinb_toy,
         celltype = celltype_toy,
         disper = 1,
         var_dropout =1,
         var_count = 1,
         delta = 1)
```

---

zinb_toy *example zinb object after fitting a toy dataset with 5 cells and 10 genes*

Description

The toy dataset is generated from the following code require(zinbwave) GeneExpr = matrix(rpois(50,1),nrow = 10,ncol = 5) rownames(GeneExpr) = paste0("gene",seq_len(nrow(GeneExpr))) colnames(GeneExpr) = paste0("cell",seq_len(ncol(GeneExpr))) celltype = as.factor(sample(c(1,2,3),5,replace = TRUE)) zinb = zinbFit(Y = GeneExpr,K=2)

Usage

```
data(zinb_toy)
```

Index

∗ **data**
    celltype, 4
    celltype_toy, 4
    disperPlot, 6
    exprdata, 6
    zinb_toy, 16
∗ **export**
    BinaryPCA, 2
    diagnose, 5
    getGeneExpr, 7
    getLoading, 7
    getScore, 8
    gradient, 8
    gradient_chunk, 9
    scBFA, 13
    scNoiseSim, 15
∗ **internal**
    InitBinaryFA, 9
    neg_loglikelihood, 10
    neg_loglikelihood_chunk, 11
    OptimBFA, 11
    restore, 12

BinaryPCA, 2

celltype, 4
celltype_toy, 4

diagnose, 5
disperPlot, 6

exprdata, 6

getGeneExpr, 7
getLoading, 7
getScore, 8
gradient, 8
gradient_chunk, 9

InitBinaryFA, 9

neg_loglikelihood, 10
neg_loglikelihood_chunk, 11

OptimBFA, 11

restore, 12

scBFA, 13
scNoiseSim, 15

zinb_toy, 16