Package: scTenifoldNet (via r-universe)

August 21, 2024

Type Package

Title Construct and Compare scGRN from Single-Cell Transcriptomic Data

Version 1.3

Description A workflow based on machine learning methods to construct and compare single-cell gene regulatory networks (scGRN) using single-cell RNA-seq (scRNA-seq) data collected from different conditions. Uses principal component regression, tensor decomposition, and manifold alignment, to accurately identify even subtly shifted gene expression programs. See <doi:10.1016/j.patter.2020.100139> for more details.

URL https://github.com/cailab-tamu/scTenifoldNet

BugReports https://github.com/cailab-tamu/scTenifoldNet/issues

License GPL (>=2)

Encoding UTF-8

LazyData true

RoxygenNote 7.1.2

Imports pbapply, RSpectra, Matrix, methods, stats, utils, MASS, RhpcBLASctl

Suggests testthat (>= 2.1.0)

Repository https://cailab-tamu.r-universe.dev

RemoteUrl https://github.com/cailab-tamu/sctenifoldnet

RemoteRef HEAD

RemoteSha de53dc2b2f35a71f19da5cc67794b47dd8fe2c1f

Contents

`cpDecomposition` *Canonical Polyadic Decomposition*

Description

Canonical Polyadic (CP) decomposition of a tensor, aka CANDECOMP/PARAFRAC. Approximate a K-Tensor using a sum of `num_components` rank-1 K-Tensors. A rank-1 K-Tensor can be written as an outer product of K vectors. There are a total of `num_compoents *tnsr@num_modes` vectors in the output, stored in `tnsr@num_modes` matrices, each with `num_components` columns. This is an iterative algorithm, with two possible stopping conditions: either relative error in Frobenius norm has gotten below `tol`, or the `max_iter` number of iterations has been reached. For more details on CP decomposition, consult Kolda and Bader (2009).

Usage

```
cpDecomposition(tnsr, num_components = NULL, max_iter = 25, tol = 1e-05)
```

Arguments

`tnsr`	Tensor with K modes
`num_components`	the number of rank-1 K-Tensors to use in approximation
`max_iter`	maximum number of iterations if error stays above `tol`
`tol`	relative Frobenius norm error tolerance

Details

Uses the Alternating Least Squares (ALS) estimation procedure. A progress bar is included to help monitor operations on large tensors.

Value

a list containing the following

`lambdas` a vector of normalizing constants, one for each component

`U` a list of matrices - one for each mode - each matrix with `num_components` columns

`conv` whether or not `resid < tol` by the last iteration

`norm_percent` the percent of Frobenius norm explained by the approximation

`est` estimate of `tnsr` after compression

`fnorm_resid` the Frobenius norm of the error `fnorm(est-tnsr)`

`all_resids` vector containing the Frobenius norm of error for all the iterations

References

T. Kolda, B. Bader, "Tensor decomposition and applications". SIAM Applied Mathematics and Applications 2009.

`cpmNormalization` *Performs counts per million (CPM) data normalization*

Description

This function normalizes the count data present in a given matrix using counts per million normalization (CPM). Each gene count for each cell is divided by the total counts for that cell and multiplied by 1e6. No log-transformation is applied.

Usage

```
cpmNormalization(X)
```

Arguments

`X`	Raw counts matrix with cells as columns and genes (symbols) as rows

Value

A dgCMatrix object with the count per million (CPM) normalized values.

References

Vallejos, Catalina A., et al. "Normalizing single-cell RNA sequencing data: challenges and opportunities." Nature methods 14.6 (2017): 565.

Examples

```
library(scTenifoldNet)

# Simulating of a dataset following a negative binomial distribution with high sparcity (~67%)
nCells = 2000
nGenes = 100
set.seed(1)
X <- rnbinom(n = nGenes * nCells, size = 20, prob = 0.98)
X <- round(X)
X <- matrix(X, ncol = nCells)
rownames(X) <- c(paste0('ng', 1:90), paste0('mt-', 1:10))

# Performing Single cell quality control
qcOutput <- scQC(
  X = X,
  minLibSize = 30,
  removeOutlierCells = TRUE,
  minPCT = 0.05,
```

```
  maxMTratio = 0.1
)
# Performing Counts per million Normalization (CPM)
normalizationOutput <- cpmNormalization(qcOutput)

# Visualizing the differences
oldPar <- par(no.readonly = TRUE)

par(
  mfrow = c(1, 2),
  mar = c(3, 3, 1, 1),
  mgp = c(1.5, 0.5, 0)
)
plot(
  Matrix::colSums(qcOutput),
  ylab = 'Library Size',
  xlab = 'Cell',
  main = 'Before CPM Normalization'
)
plot(
  Matrix::colSums(normalizationOutput),
  ylab = 'Library Size',
  xlab = 'Cell',
  main = 'After CPM Normalization'
)

par(oldPar)
```

`dRegulation` *Evaluates gene differential regulation based on manifold alignment distances.*

Description

Using the output of the non-linear manifold alignment, this function computes the Euclidean distance between the coordinates for the same gene in both conditions. Calculated distances are then transformed using Box-Cox power transformation, and standardized to ensure normality. P-values are assigned following the chi-square distribution over the fold-change of the squared distance computed with respect to the expectation.

Usage

```
dRegulation(manifoldOutput)
```

Arguments

`manifoldOutput`	A matrix. The output of the non-linear manifold alignment, a labeled matrix with two times the number of shared genes as rows (X_ genes followed by Y_ genes in the same order) and `d` number of columns.

Value

A data frame with 6 columns as follows:

- `gene` A character vector with the gene id identified from the `manifoldAlignment` output.
- `distance` A numeric vector of the Euclidean distance computed between the coordinates of the same gene in both conditions.
- `Z` A numeric vector of the Z-scores computed after Box-Cox power transformation.
- `FC` A numeric vector of the FC computed with respect to the expectation.
- `p.value` A numeric vector of the p-values associated to the fold-changes, probabilities are asigned as $P[X > x]$ using the Chi-square distribution with one degree of freedom.
- `p.adj` A numeric vector of adjusted p-values using Benjamini & Hochberg (1995) FDR correction.

References

- Benjamini, Y., and Yekutieli, D. (2001). The control of the false discovery rate in multiple testing under dependency. Annals of Statistics, 29, 1165-1188. doi: 10.1214/aos/1013699998.

Examples

```
library(scTenifoldNet)

# Simulating of a dataset following a negative binomial distribution with high sparcity (~67%)
nCells = 2000
nGenes = 100
set.seed(1)
X <- rnbinom(n = nGenes * nCells, size = 20, prob = 0.98)
X <- round(X)
X <- matrix(X, ncol = nCells)
rownames(X) <- c(paste0('ng', 1:90), paste0('mt-', 1:10))

# Performing Single cell quality control
qcOutput <- scQC(
  X = X,
  minLibSize = 30,
  removeOutlierCells = TRUE,
  minPCT = 0.05,
  maxMTratio = 0.1
)

# Computing 3 single-cell gene regulatory networks each one from a subsample of 500 cells
xNetworks <- makeNetworks(X = qcOutput,
                          nNet = 3,
                          nCells = 500,
                          nComp = 3,
                          scaleScores = TRUE,
                          symmetric = FALSE,
                          q = 0.95
                          )
```

```
# Computing a K = 3 CANDECOMP/PARAFAC (CP) Tensor Decomposition
tdOutput <- tensorDecomposition(xNetworks, K = 3, maxError = 1e5, maxIter = 1e3)

## Not run:
# Computing the alignment
# For this example, we are using the same input, the match should be perfect.
maOutput <- manifoldAlignment(tdOutput$X, tdOutput$X)

# Evaluating the difference in regulation
dcOutput <- dRegulation(maOutput, minFC = 0)
head(dcOutput)

# Plotting
# If FDR < 0.05, the gene will be colored in red.
geneColor <- ifelse(dcOutput$p.adj < 0.05, 'red', 'black')
qqnorm(dcOutput$Z, main = 'Standardized Distance', pch = 16, col = geneColor)
qqline(dcOutput$Z)

## End(Not run)
```

`makeNetworks` *Computes gene regulatory networks for subsamples of cells based on principal component regression.*

Description

This function computes `nNet` gene regulatory networks for a randomly selected subsample of `nCells` cells based on principal component regression (PCR), a technique based on principal component analysis. In PCR, the outcome variable is regressed over a `nComp` number of for principal components computed from a set of covariates to estimate the unknown regression coefficients in the model. `pcNet` function computes the PCR coefficients for each gene one at a time using all the others as covariates, to construct an all by all gene regulatory network.

Usage

```
makeNetworks(
  X,
  nNet = 10,
  nCells = 500,
  nComp = 3,
  scaleScores = TRUE,
  symmetric = FALSE,
  q = 0.95,
  nCores = parallel::detectCores()
)
```

Arguments

X	A filtered and normlized gene expression matrix with cells as columns and genes as rows.
nNet	An integer value. The number of networks based on principal components regression to generate.
nCells	An integer value. The number of cells to subsample each time to generate a network.
nComp	An integer value. The number of principal components in PCA to generate the networks. Should be greater than 2 and lower than the total number of genes.
scaleScores	A boolean value (TRUE/FALSE), if TRUE, the weights will be normalized such that the maximum absolute value is 1.
symmetric	A boolean value (TRUE/FALSE), if TRUE, the weights matrix returned will be symmetric.
q	A decimal value between 0 and 1. Represent the cut-off threshold of top q% relationships to be returned.
nCores	An integer value. Defines the number of cores to be used.

Details

Principal component regression may be broadly divided into three major steps:

1. Perform PCA on the observed covariates data matrix to obtain nComp number of the principal components.
2. Regress the observed vector of outcomes on the selected principal components as covariates, using ordinary least squares regression to get a vector of estimated regression coefficients
3. Transform this vector back to the scale of the actual covariates, using the eigenvectors corresponding to the selected principal components to get the final PCR estimator for estimating the regression coefficients characterizing the original model.

Value

A list with nNet gene regulatory networks in dgCMatrix format. Each one computed from a randomly selected subsample of nCells cells.

References

- Gill, Ryan, Somnath Datta, and Susmita Datta. "dna: An R package for differential network analysis." Bioinformation 10.4 (2014): 233.
- Jolliffe, Ian T. "A note on the use of principal components in regression." Journal of the Royal Statistical Society: Series C (Applied Statistics) 31.3 (1982): 300-303.
- Massy, William F. "Principal components regression in exploratory statistical research." Journal of the American Statistical Association 60.309 (1965): 234-256.

Examples

```
library(scTenifoldNet)

# Simulating of a dataset following a negative binomial distribution with high sparcity (~67%)
nCells = 2000
nGenes = 100
set.seed(1)
X <- rnbinom(n = nGenes * nCells, size = 20, prob = 0.98)
X <- round(X)
X <- matrix(X, ncol = nCells)
rownames(X) <- c(paste0('ng', 1:90), paste0('mt-', 1:10))

# Performing Single cell quality control
qcOutput <- scQC(
  X = X,
  minLibSize = 30,
  removeOutlierCells = TRUE,
  minPCT = 0.05,
  maxMTratio = 0.1
)

# Computing 3 single-cell gene regulatory networks each one from a subsample of 500 cells
mnOutput <- makeNetworks(X = X,
                         nNet = 3,
                         nCells = 500,
                         nComp = 3,
                         scaleScores = TRUE,
                         symmetric = FALSE,
                         q = 0.95
                         )

# Verifying the class
class(mnOutput)

# Verifying the number of networks
length(mnOutput)

# Veryfying the dimention of the networks
lapply(mnOutput,dim)

# Single-cell gene regulatory networks
mnOutput[[1]][1:10,1:10]
mnOutput[[2]][1:10,1:10]
mnOutput[[3]][1:10,1:10]
```

`manifoldAlignment` *Performs non-linear manifold alignment of two gene regulatory networks.*

Description

Build comparable low-dimensional features for two weight-averaged denoised single-cell gene regulatory networks. Using a non-linear network embedding method `manifoldAlignment` aligns two gene regulatory networks and finds the structural similarities between them. This function is a wrapper of the `Python` code provided by Vu et al., (2012) at https://github.com/all-umass/ManifoldWarping.

Usage

```
manifoldAlignment(X, Y, d = 30, nCores = parallel::detectCores())
```

Arguments

`X`	A gene regulatory network.
`Y`	A gene regulatory network.
`d`	The dimension of the low-dimensional feature space.
`nCores`	An integer value. Defines the number of cores to be used.

Details

Manifold alignment builds connections between two or more disparate data sets by aligning their underlying manifolds and provides knowledge transfer across the data sets. For further information please see: Wang et al., (2009)

Value

A low-dimensional projection for two the two gene regulatory networks used as input. The output is a labeled matrix with two times the number of shared genes as rows (X_ genes followed by Y_ genes in the same order) and `d` number of columns.

References

- Vu, Hoa Trong, Clifton Carey, and Sridhar Mahadevan. "Manifold warping: Manifold alignment over time." Twenty-Sixth AAAI Conference on Artificial Intelligence. 2012.
- Wang, Chang, and Sridhar Mahadevan. "A general framework for manifold alignment." 2009 AAAI Fall Symposium Series. 2009.

Examples

```
library(scTenifoldNet)

# Simulating of a dataset following a negative binomial distribution with high sparcity (~67%)
nCells = 2000
nGenes = 100
set.seed(1)
X <- rnbinom(n = nGenes * nCells, size = 20, prob = 0.98)
X <- round(X)
X <- matrix(X, ncol = nCells)
rownames(X) <- c(paste0('ng', 1:90), paste0('mt-', 1:10))
```

```
# Performing Single cell quality control
qcOutput <- scQC(
  X = X,
  minLibSize = 30,
  removeOutlierCells = TRUE,
  minPCT = 0.05,
  maxMTratio = 0.1
)

# Computing 3 single-cell gene regulatory networks each one from a subsample of 500 cells
xNetworks <- makeNetworks(X = X,
                          nNet = 3,
                          nCells = 500,
                          nComp = 3,
                          scaleScores = TRUE,
                          symmetric = FALSE,
                          q = 0.95
                          )

# Computing a K = 3 CANDECOMP/PARAFAC (CP) Tensor Decomposition
tdOutput <- tensorDecomposition(xNetworks, K = 3, maxError = 1e5, maxIter = 1e3)

## Not run:
# Computing the alignment
# For this example, we are using the same input, the match should be perfect.
maOutput <- manifoldAlignment(tdOutput$X, tdOutput$X)

# Separating the coordinates for each gene
X <- maOutput[grepl('X_', rownames(maOutput)),]
Y <- maOutput[grepl('Y_', rownames(maOutput)),]

# Plotting
# X Points
plot(X, pch = 16)

# Y Points
points(Y, col = 'red')

# Legend
legend('topright', legend = c('X', 'Y'),
       col = c('black', 'red'), bty = 'n',
       pch = c(16,1), cex = 0.7)

## End(Not run)
```

---

pcNet *Computes a gene regulatory network based on principal component regression*

Description

This function computes a gene regulatory network based on principal component regression (PCR), a technique based on principal component analysis. In PCR, the outcome variable is regressed over a nComp number of for principal components computed from a set of covariates to estimate the unknown regression coefficients in the model. pcNet function computes the PCR coefficients for each gene one at a time using all the others as covariates, to construct an all by all gene regulatory network.

Usage

```
pcNet(
  X,
  nComp = 3,
  scaleScores = TRUE,
  symmetric = FALSE,
  q = 0,
  verbose = TRUE,
  nCores = parallel::detectCores()
)
```

Arguments

X	A filtered and normalized gene expression matrix with cells as columns and genes as rows.
nComp	An integer value. The number of principal components in PCA to generate the networks. Should be greater than 2 and lower than the total number of genes.
scaleScores	A boolean value (TRUE/FALSE), if TRUE, the weights will be normalized such that the maximum absolute value is 1.
symmetric	A boolean value (TRUE/FALSE), if TRUE, the weights matrix returned will be symmetric.
q	A decimal value between 0 and 1. Defines the cut-off threshold of top q% relationships to be returned.
verbose	A boolean value (TRUE/FALSE), if TRUE, a progress bar is shown.
nCores	An integer value. Defines the number of cores to be used.

Details

Principal component regression may be broadly divided into three major steps:

1. Perform PCA on the observed covariates data matrix to obtain nComp number of the principal components.
2. Regress the observed vector of outcomes on the selected principal components as covariates, using ordinary least squares regression to get a vector of estimated regression coefficients
3. Transform this vector back to the scale of the actual covariates, using the eigenvectors corresponding to the selected principal components to get the final PCR estimator for estimating the regression coefficients characterizing the original model.

Value

A gene regulatory network in dgCMatrix format.

References

- Gill, Ryan, Somnath Datta, and Susmita Datta. "dna: An R package for differential network analysis." Bioinformation 10.4 (2014): 233.
- Jolliffe, Ian T. "A note on the use of principal components in regression." Journal of the Royal Statistical Society: Series C (Applied Statistics) 31.3 (1982): 300-303.
- Massy, William F. "Principal components regression in exploratory statistical research." Journal of the American Statistical Association 60.309 (1965): 234-256.

Examples

```
library(scTenifoldNet)

# Simulating of a dataset following a negative binomial distribution with high sparcity (~67%)
nCells = 2000
nGenes = 100
set.seed(1)
X <- rnbinom(n = nGenes * nCells, size = 20, prob = 0.98)
X <- round(X)
X <- matrix(X, ncol = nCells)
rownames(X) <- c(paste0('ng', 1:90), paste0('mt-', 1:10))

# Performing Single cell quality control
qcOutput <- scQC(
  X = X,
  minLibSize = 30,
  removeOutlierCells = TRUE,
  minPCT = 0.05,
  maxMTratio = 0.1
)

# Computing a single-cell gene regulatory network using principal component regression
# Non-symmetric
pcnetOutput <- pcNet(X = qcOutput, nComp = 3, scaleScores = TRUE, symmetric = FALSE, q = 0)
pcnetOutput[1:10,1:10]

# Symmetric
pcnetOutput <- pcNet(X = qcOutput, nComp = 3, scaleScores = TRUE, symmetric = TRUE, q = 0)
pcnetOutput[1:5,1:5]
```

---

scQC *Performs single-cell data quality control*

Description

This function performs quality control filters over the provided input matrix, the function checks for minimum cell library size, mitochondrial ratio, outlier cells, and the fraction of cells where a gene is expressed.

Usage

```
scQC(
  X,
  minLibSize = 1000,
  removeOutlierCells = TRUE,
  minPCT = 0.05,
  maxMTratio = 0.1
)
```

Arguments

X	Raw counts matrix with cells as columns and genes (symbols) as rows.
minLibSize	An integer value. Defines the minimum library size required for a cell to be included in the analysis.
removeOutlierCells	A boolean value (TRUE/FALSE), if TRUE, the identified cells with library size greater than 1.58 IQR/sqrt(n) computed from the sample, are removed. For further details see: ?boxplot.stats
minPCT	A decimal value between 0 and 1. Defines the minimum fraction of cells where the gene needs to be expressed to be included in the analysis.
maxMTratio	A decimal value between 0 and 1. Defines the maximum ratio of mitochondrial reads (mithocondrial reads / library size) present in a cell to be included in the analysis. It's computed using the symbol genes starting with 'MT-' non-case sensitive.

Value

A dgCMatrix object with the cells and the genes that pass the quality control filters.

References

Ilicic, Tomislav, et al. "Classification of low quality cells from single-cell RNA-seq data." Genome biology 17.1 (2016): 29.

Examples

```
library(scTenifoldNet)

# Simulating of a dataset following a negative binomial distribution with high sparcity (~67%)
nCells = 2000
nGenes = 100
set.seed(1)
```

```
X <- rnbinom(n = nGenes * nCells, size = 20, prob = 0.98)
X <- round(X)
X <- matrix(X, ncol = nCells)
rownames(X) <- c(paste0('ng', 1:90), paste0('mt-', 1:10))

# Performing Single cell quality control
qcOutput <- scQC(
  X = X,
  minLibSize = 30,
  removeOutlierCells = TRUE,
  minPCT = 0.05,
  maxMTratio = 0.1
)

# Visualizing the Differences
oldPar <- par(no.readonly = TRUE)

par(
  mfrow = c(2, 2),
  mar = c(3, 3, 1, 1),
  mgp = c(1.5, 0.5, 0)
)
# Library Size
plot(
  Matrix::colSums(X),
  ylim = c(20, 70),
  ylab = 'Library Size',
 xlab = 'Cell',
  main = 'Library Size - Before QC'
)
abline(h = c(30, 58),
       lty = 2,
       col = 'red')
plot(
  Matrix::colSums(qcOutput),
  ylim = c(20, 70),
  ylab = 'Library Size',
  xlab = 'Cell',
  main = 'Library Size - After QC'
)
abline(h = c(30, 58),
       lty = 2,
       col = 'red')
# Mitochondrial ratio
mtGenes <- grepl('^mt-', rownames(X), ignore.case = TRUE)
plot(
  Matrix::colSums(X[mtGenes,]) / Matrix::colSums(X),
  ylim = c(0, 0.3),
  ylab = 'Mitochondrial Ratio',
  xlab = 'Cell',
  main = 'Mitochondrial Ratio - Before QC'
)
abline(h = c(0.1), lty = 2, col = 'red')
```

```
plot(
  Matrix::colSums(qcOutput[mtGenes,]) / Matrix::colSums(qcOutput),
  ylim = c(0, 0.3),
  ylab = 'Mitochondrial Ratio',
  xlab = 'Cell',
  main = 'Mitochondrial Ratio - Before QC'
)
abline(h = c(0.1), lty = 2, col = 'red')

par(oldPar)
```

---

scTenifoldNet *scTenifoldNet*

Description

Construct and compare single-cell gene regulatory networks (scGRNs) using single-cell RNA-seq (scRNA-seq) data sets collected from different conditions based on principal component regression, tensor decomposition, and manifold alignment.

Usage

```
scTenifoldNet(
  X,
  Y,
  qc = TRUE,
  qc_minLibSize = 1000,
  qc_removeOutlierCells = TRUE,
  qc_minPCT = 0.05,
  qc_maxMTratio = 0.1,
  nc_nNet = 10,
  nc_nCells = 500,
  nc_nComp = 3,
  nc_symmetric = FALSE,
  nc_scaleScores = TRUE,
  nc_q = 0.05,
  td_K = 3,
  td_nDecimal = 1,
  td_maxIter = 1000,
  td_maxError = 1e-05,
  ma_nDim = 30,
  nCores = parallel::detectCores()
)
```

Arguments

X	Raw counts matrix with cells as columns and genes (symbols) as rows.
Y	Raw counts matrix with cells as columns and genes (symbols) as rows.

qc	A boolean value (TRUE/FALSE), if TRUE, a quality control is applied over the data.
qc_minLibSize	An integer value. Defines the minimum library size required for a cell to be included in the analysis.
qc_removeOutlierCells	A boolean value (TRUE/FALSE), if TRUE, the identified cells with library size greater than 1.58 IQR/sqrt(n) computed from the sample, are removed. For further details see: ?boxplot.stats
qc_minPCT	A decimal value between 0 and 1. Defines the minimum fraction of cells where the gene needs to be expressed to be included in the analysis.
qc_maxMTratio	A decimal value between 0 and 1. Defines the maximum ratio of mitochondrial reads (mithocondrial reads / library size) present in a cell to be included in the analysis. It's computed using the symbol genes starting with 'MT-' non-case sensitive.
nc_nNet	An integer value. The number of networks based on principal components regression to generate.
nc_nCells	An integer value. The number of cells to subsample each time to generate a network.
nc_nComp	An integer value. The number of principal components in PCA to generate the networks. Should be greater than 2 and lower than the total number of genes.
nc_symmetric	A boolean value (TRUE/FALSE), if TRUE, the weights matrix returned will be symmetric.
nc_scaleScores	A boolean value (TRUE/FALSE), if TRUE, the weights will be normalized such that the maximum absolute value is 1.
nc_q	A decimal value between 0 and 1. Defines the cut-off threshold of top q% relationships to be returned.
td_K	An integer value. Defines the number of rank-one tensors used to approximate the data using CANDECOMP/PARAFAC (CP) Tensor Decomposition.
td_nDecimal	An integer value indicating the number of decimal places to be used.
td_maxIter	An integer value. Defines the maximum number of iterations if error stay above td_maxError.
td_maxError	A decimal value between 0 and 1. Defines the relative Frobenius norm error tolerance.
ma_nDim	An integer value. Defines the number of dimensions of the low-dimensional feature space to be returned from the non-linear manifold alignment.
nCores	An integer value. Defines the number of cores to be used.

Value

A list with 3 slots as follows:

- tensorNetworks: The generated weight-averaged denoised gene regulatory networks using CANDECOMP/PARAFAC (CP) Tensor Decomposition.
- manifoldAlignment: The generated low-dimensional features result of the non-linear manifold alignment.
- diffRegulation The results of the differential regulation analysis.

Examples

```
library(scTenifoldNet)

# Simulating of a dataset following a negative binomial distribution with high sparcity (~67%)
nCells = 2000
nGenes = 100
set.seed(1)
X <- rnbinom(n = nGenes * nCells, size = 20, prob = 0.98)
X <- round(X)
X <- matrix(X, ncol = nCells)
rownames(X) <- c(paste0('ng', 1:90), paste0('mt-', 1:10))

# Generating a perturbed network modifying the expression of genes 10, 2 and 3
Y <- X
Y[10,] <- Y[50,]
Y[2,] <- Y[11,]
Y[3,] <- Y[5,]

## Not run:
# scTenifoldNet
Output <- scTenifoldNet(X = X, Y = Y,
                        nc_nNet = 10, nc_nCells = 500,
                        td_K = 3, qc_minLibSize = 30)

# Structure of the output
str(Output)

# Accessing the computed weight-averaged denoised gene regulatory networks

# Network for sample X
igraph::graph_from_adjacency_matrix(adjmatrix = Output$tensorNetworks$X, weighted = TRUE)
# IGRAPH 15cbeea DNW- 100 2836 --
# + attr: name (v/c), weight (e/n)
# + edges from 15cbeea (vertex names):
#   [1] ng6 ->ng1 ng12->ng1 ng14->ng1 ng24->ng1 ng28->ng1
# [6] ng31->ng1 ng42->ng1 ng44->ng1 ng49->ng1 ng55->ng1
# [11] ng56->ng1 ng59->ng1 ng62->ng1 ng63->ng1 ng72->ng1
# [16] ng73->ng1 ng74->ng1 ng77->ng1 ng80->ng1 ng82->ng1
# [21] ng83->ng1 ng87->ng1 ng89->ng1 mt-1->ng1 mt-5->ng1
# [26] mt-7->ng1 ng27->ng3 ng28->ng3 ng31->ng3 ng32->ng3
# [31] ng44->ng3 ng59->ng3 ng62->ng3 ng72->ng3 ng73->ng3
# [36] ng74->ng3 ng77->ng3 ng82->ng3 ng87->ng3 ng89->ng3
# + ... omitted several edges

# Network for sample Y
igraph::graph_from_adjacency_matrix(adjmatrix = Output$tensorNetworks$Y, weighted = TRUE)
#IGRAPH 3ad1533 DNW- 100 725 --
# + attr: name (v/c), weight (e/n)
# + edges from 3ad1533 (vertex names):
#   [1] ng2 ->ng2 ng3 ->ng2 ng5 ->ng2 ng6 ->ng2
# [5] ng7 ->ng2 ng8 ->ng2 ng9 ->ng2 ng10->ng2
# [9] ng11->ng2 ng12->ng2 ng13->ng2 ng15->ng2
```

```
# [13] ng16->ng2 ng17->ng2 ng18->ng2 ng20->ng2
# [17] ng21->ng2 ng22->ng2 ng23->ng2 ng24->ng2
# [21] ng25->ng2 ng26->ng2 ng28->ng2 ng29->ng2
# [25] ng30->ng2 ng31->ng2 ng33->ng2 ng34->ng2
# [29] ng35->ng2 ng36->ng2 ng38->ng2 ng39->ng2
# + ... omitted several edges

# Accessing the manifold alignment result

head(Output$manifoldAlignment)
#            NLMA 1      NLMA 2      NLMA 3       NLMA 4        NLMA 5
# X_ng1  0.0068499391  0.01096706  0.03077900  0.002655469 -0.0136455614
# X_ng2  0.3356288575 -0.03551752 -0.18463680 -0.193353751  0.3398606363
# X_ng3 -0.1285177133 -0.20064344  0.20926567  0.059542294 -0.0099528441
# X_ng4  0.0029881645 -0.01267593  0.01195683  0.007331123  0.0003031888
# X_ng5 -0.1192632208 -0.18475439  0.27616148  0.112944009 -0.0281827702
# X_ng6  0.0005911568  0.02557475  0.07527792 -0.191180647 -0.1165095115
#            NLMA 6      NLMA 7       NLMA 8       NLMA 9     NLMA 10
# X_ng1 -0.029852128 0.007539925  0.009299591 -0.009813157 -0.01360414
# X_ng2 -0.313361443 0.146429589  0.006286777  0.162023788 -0.04307899
# X_ng3 -0.008733285 0.172084611  0.508056218  0.199322512 -0.07935797
# X_ng4 -0.004680652 0.005344541  0.002634755 -0.003376544 -0.01100757
# X_ng5 -0.126328797 0.190769152 -0.468107666  0.170278281 -0.06744795
# X_ng6 -0.051266264 0.063822269  0.011060924 -0.134880459 -0.02579998
#           NLMA 11      NLMA 12      NLMA 13      NLMA 14      NLMA 15
# X_ng1 -0.0199528840  0.008035130  0.004631187  0.000807797  0.011960838
# X_ng2 -0.0138200390 -0.002847701 -0.004404942  0.008024704  0.006040799
# X_ng3  0.0232384468 -0.031398116 -0.007026934  0.028956700 -0.002112626
# X_ng4  0.0012864539 -0.018915289  0.003835404  0.004054159 -0.002546324
# X_ng5  0.0232899093 -0.040974531 -0.006759459  0.025415953 -0.007518957
# X_ng6 -0.0001650355  0.023277338  0.006646904 -0.002683418 -0.112688129
#          NLMA 16      NLMA 17     NLMA 18      NLMA 19       NLMA 20
# X_ng1 -0.016962988 -0.016649748  0.01140020 -0.006632691 -0.0005015655
# X_ng2  0.007543775 -0.016188689  0.02517684  0.014814415  0.0162617154
# X_ng3 -0.005598267 -0.006975026  0.05218029  0.006731063  0.0183436415
# X_ng4  0.003207934 -0.001784120  0.01093237 -0.001192860  0.0028746990
# X_ng5 -0.009555879 -0.007429166  0.05206441  0.006534604  0.0170071357
# X_ng6 -0.065437425  0.110728870 -0.12746932  0.335610531  0.1341842827
#          NLMA 21      NLMA 22      NLMA 23      NLMA 24      NLMA 25
# X_ng1  0.003113385 -0.023311350 -0.026415944 7.085995e-04  0.053898102
# X_ng2  0.001390569  0.001191301 -0.015621435 2.359703e-03 -0.013418093
# X_ng3 -0.007483171  0.011496519  0.004164546 2.764407e-02 -0.004527981
# X_ng4  0.020316634 -0.002796092  0.032119363 4.203867e-05 -0.002251366
# X_ng5 -0.004963436  0.016525449  0.009683698 2.564700e-02  0.002286340
# X_ng6  0.229199525  0.340639745 -0.041216345 3.599596e-03  0.008572652
#          NLMA 26       NLMA 27      NLMA 28     NLMA 29      NLMA 30
# X_ng1  0.065832029 -0.0080248854  0.107300843 -0.02902323 -0.005337500
# X_ng2 -0.007982259 -0.0026295392 -0.001765851  0.01491257 -0.003546343
# X_ng3  0.009770602  0.0008819272  0.014564070 -0.01568192 -0.017450667
# X_ng4  0.015802609  0.0012975576 -0.003406675 -0.01774975 -0.003300053
# X_ng5  0.003401007  0.0001761177  0.013622016 -0.01224127 -0.013909178
# X_ng6 -0.089450710 -0.0763838722 -0.107751916 -0.05841353 -0.059217012
```

```
# Differential Regulation
head(Output$diffRegulation,n = 10)
# gene    distance        Z        FC      p.value      p.adj
# 2   ng2 0.023526702 2.762449 12.193413 0.0004795855 0.02414332
# 50 ng50 0.023514429 2.761550 12.180695 0.0004828665 0.02414332
# 11 ng11 0.022443941 2.681598 11.096894 0.0008647241 0.02882414
# 3   ng3 0.020263415 2.508478  9.045415 0.0026335445 0.06583861
# 10 ng10 0.019194561 2.417929  8.116328 0.0043868326 0.07711821
# 5   ng5 0.019079975 2.407977  8.019712 0.0046270923 0.07711821
# 31 ng31 0.013632541 1.865506  4.094085 0.0430335257 0.61476465
# 96 mt-6 0.011401177 1.589757  2.863536 0.0906081350 0.90977795
# 59 ng59 0.009835354 1.368238  2.130999 0.1443466682 0.90977795
# 62 ng62 0.007995812 1.067193  1.408408 0.2353209153 0.90977795

# Plotting
# Genes with FDR < 0.1 are labeled as red
set.seed(1)
qChisq <- rchisq(100,1)
geneColor <- rev(ifelse(Output$diffRegulation$p.adj < 0.1, 10,1))
qqplot(qChisq, Output$diffRegulation$FC, pch = 16, main = 'H0', col = geneColor,
       xlab = expression(X^2~Quantiles), ylab = 'FC', xlim=c(0,8), ylim=c(0,13))
qqline(qChisq)
legend('bottomright', legend = c('FDR < 0.1'), pch = 16, col = 'red', bty='n', cex = 0.7)

## End(Not run)
```

`tensorDecomposition` *Performs CANDECOMP/PARAFAC (CP) Tensor Decomposition.*

Description

Generate weight-averaged denoised gene regulatory networks using CANDECOMP/PARAFAC (CP) Tensor Decomposition. The `tensorDecomposition` function takes one or two lists of gene regulatory matrices, if two list are provided, the shared genes are selected and the CP tensor decomposition is performed independently for each list (3d-tensor). The tensor decomposed matrices are then averaged to generate weight-averaged denoised networks.

Usage

```
tensorDecomposition(
  xList,
  yList = NULL,
  nDecimal = 1,
  K = 5,
  maxError = 1e-05,
  maxIter = 1000
)
```

Arguments

xList	A list of gene regulatory networks.
yList	Optional. A list of gene regulatory networks.
nDecimal	An integer value indicating the number of decimal places to be used.
K	The number of rank-one tensors used to approximate the data using CANDECOMP/PARAFAC (CP) Tensor Decomposition,
maxError	A decimal value between 0 and 1. Defines the relative Frobenius norm error tolerance
maxIter	An integer value. Defines the maximum number of iterations if error stay above maxError.

Details

CANDECOMP/PARAFRAC (CP) tensor decomposition approximate a K-Tensor using a sum of K rank-1 K-Tensors. A rank-1 K-Tensor can be written as an outer product of K vectors. This is an iterative algorithm, with two possible stopping conditions: either relative error in Frobenius norm has gotten below maxError, or the maxIter number of iterations has been reached. For more details on CP decomposition, consult Kolda and Bader (2009) and Morup (2011).

Value

A list of weight-averaged denoised gene regulatory networks.

Author(s)

This is an adaptation of the code provided by Li, J., Bien, J., & Wells, M. T. (2018)

References

- Li, J., Bien, J., & Wells, M. T. (2018). rTensor: An R Package for Multidimensional Array (Tensor) Unfolding, Multiplication, and Decomposition. Journal of Statistical Software, 87(10), 1-31.
- Kolda, Tamara G., and Brett W. Bader. "Tensor decompositions and applications." SIAM review 51.3 (2009): 455-500.
- Morup, Morten. "Applications of tensor (multiway array) factorizations and decompositions in data mining." Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 1.1 (2011): 24-40.

Examples

```
library(scTenifoldNet)

# Simulating of a dataset following a negative binomial distribution with high sparcity (~67%)
nCells = 2000
nGenes = 100
set.seed(1)
X <- rnbinom(n = nGenes * nCells, size = 20, prob = 0.98)
```

```
X <- round(X)
X <- matrix(X, ncol = nCells)
rownames(X) <- c(paste0('ng', 1:90), paste0('mt-', 1:10))

# Performing Single cell quality control
qcOutput <- scQC(
  X = X,
  minLibSize = 30,
  removeOutlierCells = TRUE,
  minPCT = 0.05,
  maxMTratio = 0.1
)

# Computing 3 single-cell gene regulatory networks each one from a subsample of 500 cells
mnOutput <- makeNetworks(X = X,
                         nNet = 3,
                         nCells = 500,
                         nComp = 3,
                         scaleScores = TRUE,
                         symmetric = FALSE,
                         q = 0.95
                         )

# Computing a K = 3 CANDECOMP/PARAFAC (CP) Tensor Decomposition
tdOutput <- tensorDecomposition(mnOutput, K = 3, maxError = 1e5, maxIter = 1e3)

# Verifying the number of networks
length(tdOutput)

# Veryfying the dimention of the networks
lapply(tdOutput,dim)

# Weight-averaged denoised single-cell gene regulatory networks
tdOutput[[1]][1:10,1:10]
```

Index