Package 'BioTIP'

April 15, 2020

Type Package

Title BioTIP: An R package for characterization of Biological Tipping-Point

Version 1.0.0

Author Zhezhen Wang, Biniam Feleke, Qier An, Antonio Feliciano y Pleyto and Xinan Yang

Maintainer
Zhezhen Wang <zhezhen@uchicago.edu>, X Holly Yang < xyang2@uchicago> and Yuxi (Jennifer) Sun <ysun11@uchicago.edu>

Description Adopting tipping-point theory to transcriptome profiles to unravel disease regulatory trajectory.

Depends R (>= 3.6)

License GPL-2

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 6.1.1

Suggests knitr, markdown, base, rmarkdown, ggplot2

VignetteBuilder knitr

biocViews Sequencing, RNASeq, GeneExpression, Transcription, Software

URL https://github.com/xyang2uchicago/BioTIP

Imports igraph, cluster, psych, stringr, GenomicRanges

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

git_branch RELEASE_3_10

git_last_commit cb9a58f

git_last_commit_date 2019-10-29

Date/Publication 2020-04-14

R topics documented:

`cod` *cod dataset*

Description

A subset of gencode_gr extracted as: cod <- subset(gencode_gr, biotype == 'protein_coding')

Usage

```
cod
```

Format

A dataframe with 3 data columns and 1 metadata column.

seqnames chromosome names (chr21)

ranges rangeschromosome ranges on the genome (10906201-11029719)

strand specific strand of the genomic location (+,-,*)

gencode — *A chr21 data from GENCODE GRCh37*

Description

A reference human GRCh37 genome dataset from GENCODE. https://www.gencodegenes.org/human/release_25lift37.html. Included variables are as follows:

Usage

```
gencode
```

Format

A data frame of chr21 with 2619444 rows and 9 variables:

Rows Rows, include chromosome numbers

Columns Columns include seqname, source, feature, start , end, score, strand, frame and attribute

cod_gr cod_gr is a subset of sample data 'gencode_gr'

getBiotypes — *Assigning Transcript Biotypes*

Description

The purpose of the `getBiotypes()` function is to class both coding and noncoding transcripts into biotypes using the most recent GENCODE annotations. This tool can also be used to define potential lncRNAs, given an available genome transcriptome assembly (a gtf file) or any genomic loci of interest.

Usage

```
getBiotypes(full_gr, gencode_gr, intron_gr = NULL, minoverlap = 1L)
```

Arguments

`full_gr`	A GRanges object which contains either coding or noncoding transcripts. Each GRanges objects' columns requires a unique identifications. For further details refer to the GRanges package.
`gencode_gr`	A GRanges object contataining a human Chr21 GENCODE reference annotation. A metadata column, "biotype", describes the transcript type.
`intron_gr`	A GRanges object containing the coordinates of non-coding transcripts.
`minoverlap`	An IRanges argument which detects minimum overlap between two IRanges objects. For more information about minoverlap argument refer to the IRanges package.

Details

For details of findOverlaps, type.partialOverlap, type.50Overlap type.toPlot, queryhits, and subjecthits see GenomicRanges `https://www.bioconductor.org/packages/release/bioc/html/GenomicRanges.html`, IRanges `https://www.bioconductor.org/packages/release/bioc/html/IRanges.html`, and BiocManager `http://bioconductor.org/install/index.html`.

Value

A GRanges object that returns classified transcriptome biotypes.

Note

Replace the PATH_FILE when loading your data locally.

Author(s)

Zhezhen Wang and Biniam Feleke

Source

Reference GRCh37 genome `https://www.gencodegenes.org/human/release_25lift37.html` for details on gtf format visit ensemble `https://useast.ensembl.org/info/website/upload/gff.html`

References

Wang, Z. Z., J. M. Cunningham and X. H. Yang (2018).'CisPi: a transcriptomic score for disclosing cis-acting disease-associated lincRNAs.' Bioinformatics 34(17): 664-670', PMID: 30423099'

Examples

```
# Input datasets from our package's data folder
library(GenomicRanges)
data("gencode")
data("intron")
data("ILEF")

# Converting datasets to GRanges object
gencode_gr = GRanges(gencode)
ILEF_gr = GRanges(ILEF)
cod_gr = GRanges(cod)
intron_gr= GRanges(intron)

# Filtering non-coding transcripts
getBiotypes(ILEF_gr, gencode_gr, intron_gr)

## Not run: getBiotypes(intron_gr)
```

`getCluster_methods` *Clustering Network Nodes*

Description

This function runs over all states which are grouped samples. For each state, this function splits the correlation network generated from the function `getNetwork` into several sub-networks (which we called 'module'). The network nodes will be defined by the end-user. For transcriptome analysis, network nodes can be the expressed transcripts. The outputs of this function include the module IDs and node IDs per module.

Usage

```
getCluster_methods(igraphL, method = c("rw", "hcm", "km", "pam",
  "natural"), cutoff = NULL)
```

Arguments

`igraphL`	A list of numerical matrices or a list of igraph objects. The list of igraph objects can be the output from the getNetwork function.
`method`	A mathematical clustering model for analyzing network nodes. Default is a random walk ('rw'). A method could be 'rw', 'hcm', 'km', 'pam', or 'natural', where: • rw: random walk using cluster_walktrap function in igraph package. 'igraphL' has to be a list of igraph. • hcm: hierarchical clustering using function hclust) and dist, using method 'complete'. • km and pam: k-medoids or PAM algorithm using KMedoids. • natrual: if nodes are disconnected, they may naturally cluster and form sub-networks.
`cutoff`	A numeric value, default is NULL. For each method it means: • rw: the number of steps needed, see cluster_walktrap for more detail. If "cutoff" is not assigned, default of 4 will be used. • hcm, km and pam: number of clusters wanted. No default assigned. • natural: does not use this parameter.

Value

When method=rw: A list of `communities` objects of R package igraph, whose length is the length of the input object `igraphL`. These `communities` objects can be used for visualization when being assigned to the 'mark.groups' parameter of the `plot.igraph` function of the igraph package. Otherwise this function returns a list of vectors, whose length is the length of the input object `igraphL`. The names of each vector are the pre-selected transcript IDs by th function `sd_selection`. Each vector, whose length is the number of pre-selected transcript in a state, contains the module IDs.

Author(s)

Zhezhen Wang <`zhezhen@uchicago.edu`>

Examples

```
test = list('state1' = matrix(sample(1:10,6),3,3),'state2' =
matrix(sample(1:10,6),3,3),'state3' = matrix(sample(1:10,6),3,3))
#assign colnames and rownames to the matrix

for(i in names(test)){
colnames(test[[i]]) = 1:3
row.names(test[[i]]) = 1:3}

#using 'rw' or 'natural' method
igraphL <- getNetwork(test, fdr=1)
#[1] "state1:3 nodes"
#[1] "state2:3 nodes"
#[1] "state3:3 nodes"

cl <- getCluster_methods(igraphL)

#using 'km', 'pam' or 'hcm'
cl <- getCluster_methods(test, method = 'pam', cutoff=2)
```

---

getCTS *Obtain the identified BioTiP and its length*

Description

getCTS obtains the identified BioTiP and its length based off of MCI scores.

Usage

```
getCTS(maxMCI, maxMCIms)
```

Arguments

`maxMCI`	A numeric vector, whose length is the number of states. This parameter is the maximum MCI score of each state, and it can be obtained from the output of `getMaxStats`. Names need to be included in names of `maxMCIms`.
`maxMCIms`	A list of character vectors per state. The vectors are network nodes (e.g. transcript ids). This parameter is the second element of the output of the function `getMaxMCImember`.

Value

A character vector, in which the elements are the unique IDs of the network nodes of the BioTiP.

Author(s)

Antonio Feliciano y Pleyto and Zhezhen Wang <zhezhen@uchicago.edu>

Examples

```
maxMCI <- c(a = 2.56, b = 8.52, c = 2.36, d = 4.81, e = 5.26)
maxMCIms <- list(a = c("A100", "A293", "C403"), b = c("B853", "D826", "A406"), c = c("J198", "D103", "B105"), d =
identical(names(maxMCI), names(maxMCIms))
# TRUE
getCTS(maxMCI, maxMCIms)
# "Length: 3"
# "B853" "D826" "A406"
```

`getIc` *get Ic score*

Description

retreive Ic scores (Pearson correlation of genes / Pearson correlation of samples) for the identified BioTiP

Usage

```
getIc(counts, sampleL, genes, output = c("Ic", "PCCg", "PCCs"))
```

Arguments

`counts`	A numeric matrix or data frame. The rows and columns represent unique transcript IDs (geneID) and sample names, respectively.
`sampleL`	A list of vectors, whose length is the number of states. Each vector gives the sample names in a state. Note that the vector s (sample names) has to be among the column names of the R object 'df'.
`genes`	A character vector consisting of unique BioTiP ids. This can be obtained from `getMaxMCImember`
`output`	A string. Please select from 'Ic', 'PCCg', or 'PCCs'. Uses 'Ic' by default. 'PCCg' is the PCC between genes (numerator) and 'PCCs' is PCC between samples (denominator)

Value

A list of numeric values, whose length and names are inherited from `sampleL`

Author(s)

Zhezhen Wang <zhezhen@uchicago.edu>

Examples

```
counts = matrix(sample(1:100,27),3,9)
colnames(counts) = 1:9
row.names(counts) = c('loci1','loci2','loci3')
cli = cbind(1:9,rep(c('state1','state2','state3'),each = 3))
colnames(cli) = c('samples','group')
samplesL <- split(cli[,1],f = cli[,'group'])
BioTiP = c('loci1','loci2')
Ic = getIc(counts,samplesL,BioTiP)
```

getMaxMCImember *Identifying the 'Biomodule'*

Description

This function reports the 'biomodule', which is the module with the maximum Module Critical Index (MCI) scores for each state. Each state can have multiple modules (groups of subnetworks derived from the function `getCluster_methods`). This function runs over all states.

Usage

```
getMaxMCImember(membersL, MCIl, minsize = 1)
```

Arguments

`membersL`	A list of integer vectors with unique ids as names. Each vector represents the cluster number assign to that unique id. The length of this list is equal to the number of states in the study. This can be the first element of the output from function `getMCI` or the output from `getCluster_methods`, see Examples for more detail.
`MCIl`	A list of numeric vectors with unique cluster numbers as names. Each vector represents the MCI scores of that module. This can be the second element of the output from function `getMCI`.
`minsize`	A numerical value of the minimum module size (the number of transcripts in a cluster) to output for downstream analysis.

Value

A nested list whose length is the length of the input object `membersL`. Each internal list contains two objects: one object is the vector of biomodule IDs across states, and the other object is a list of transcript IDs (each defines the biomodule per state) across states.

Author(s)

Zhezhen Wang <zhezhen@uchicago.edu>

Examples

```
#1st option: get the input directly from getMCI function
test = list('state1' = matrix(sample(1:10,6),4,3),'state2' = matrix(sample(1:10,6),4,3),'state3' = matrix(sam

# assign colnames and rownames to the matrix
for(i in names(test)){
  colnames(test[[i]]) = 1:3
    row.names(test[[i]]) = c('g1','g2','g3','g4')}

cluster = list(c(1,2,2,1),c(1,2,3,1),c(2,2,1,1))
names(cluster) = names(test)
for(i in names(cluster)){
  names(cluster[[i]]) = c('g1','g2','g3','g4')}

membersL_noweight <- getMCI(cluster,test)
```

```
maxMCIms <- getMaxMCImember(membersL_noweight[[1]], membersL_noweight[[2]], min =3)
#The same as
maxMCIms <- getMaxMCImember(cluster, membersL_noweight[[2]], min =2)

## case1: using 'rw' method by default
igraphL <- getNetwork(test, fdr=1)
cl <- getCluster_methods(igraphL)
## make sure every element in list cl is a \\code{communities} object
sapply(cl,class)
##       state1        state2        state3
##"communities" "communities" "communities"

## if there is(are) state(s) that is(are) empty which will not be a communities object(s), please manually remov
cl = cl[which(sapply(cl,class) == 'communities')]

## and then run
library(igraph)
cluster = lapply(cl, membership)
maxCIms <- getMaxMCImember(cluster, membersL_noweight[[2]], min =2)

## or run function 'getMCI' and use the 1st option
membersL_noweight <- getMCI(cl,test)

## case2: using methods other than the default
cl <- getCluster_methods(test,method = "pam",cutoff = 2)

## check to make sure membersL_noweight[[2]] has values and run
maxCIms <- getMaxMCImember(cl, membersL_noweight[[2]], min =2)
```

`getMaxStats` *Get the cluster index and network nodes of biomodule*

Description

This function retrives the cluster index and network-node ids for the identified biomodule (that shows the maximum MCI score) at each state in the study.

Usage

```
getMaxStats(membersL, idx)
```

Arguments

`membersL`	A two-layer nested list of character or numeric values, any one out of the five elements outputed by the function `getMCI`.
`idx`	A vector of integers that are cluster ids of the biomodule (the module with the highest MCI score) per state. This is the first element of the result from `getMaxMCImember`

Value

A list describing the biomodule of each state, corresponding to one of the five elements (members, MCI, Sd, PCC, and PCCo) outputted by the function `getMCI`. The calss of the vector depends on the class of the input parameter `membersL`.

Author(s)

Zhezhen Wang <zhezhen@uchicago.edu>

See Also

getMCI

Examples

```
test = list('state1' = matrix(sample(1:10,6),4,3),'state2' = matrix(sample(1:10,6),4,3),'state3' = matrix(sam
# assign colnames and rownames to the matrix
for(i in names(test)){
 colnames(test[[i]]) = 1:3
 row.names(test[[i]]) = c('g1','g2','g3','g4')
}

cluster = list(c(1,2,2,1),c(1,2,3,1),c(2,2,1,1))
names(cluster) = names(test)
for(i in names(cluster)){
 names(cluster[[i]]) = c('g1','g2','g3','g4')
}
membersL_noweight <- getMCI(cluster,test)
idx = c(1,2,1)
names(idx) = names(membersL_noweight[['sd']])
selectedSD = getMaxStats(membersL_noweight[['sd']],idx)
```

---

getMCI *Calculating MCI Scores*

Description

This function calculates a module critical index (MCI) score for each module per state within a dataset. Each module is a cluster of transcripts generated from the function getCluster_methods. Note that a dataset should contains three or more states (samples in groups).

Usage

```
getMCI(groups, countsL, adjust.size = FALSE)
```

Arguments

groups	A list of elements whose length is the member of states. The elements could be either be vectors or communities object of the R package igraph. If a vector, it is the output of the function getCluster_methods. The names of each vector are the pre-selected transcript IDs generated by the function sd_selection. Each vector, whose length is the number of pre-selected transcripts in a state, contains the module IDs. If a communities object, it can be obtained by getCluster_methods using the "rw" method. It is also an output of the function sd_selection.
countsL	A list of x numeric count matrices or x data frame, where x is the number of states.
adjust.size	A boolean value indicating if MCI score should be adjusted by module size (the number of transcripts in the module) or not. Default FALSE.

Value

A list of five elements (members, MCI, Sd, PCC, and PCCo). Each of element is a two-layer nested list whose length is the length of the input object `groups`. Each internal nested list is structured according to the number of modules identified in that state.

- members: vectors of unique ids
- MCI: the MCI score
- sd: standard deviation
- PCC: Mean of pairwised Pearson Correlation Coefficient calculated among the loci in a module.
- PCCo: Mean of pairwised Pearson Correlation Coefficient calculated between the loci in a module and the loci outside that module but inside the same state.

Author(s)

Zhezhen Wang <`zhezhen@uchicago.edu`>

Examples

```
test = list('state1' = matrix(sample(1:10,6),4,3),'state2' =
matrix(sample(1:10,6),4,3),'state3' = matrix(sample(1:10,6),4,3))

#assign colnames and rownames to the matrix
for(i in names(test)){
   colnames(test[[i]]) = 1:3
   row.names(test[[i]]) = c('g1','g2','g3','g4')}

cluster = list(c(1,2,2,1),c(1,2,3,1),c(2,2,1,1))
names(cluster) = names(test)
for(i in names(cluster)){
   names(cluster[[i]]) = c('g1','g2','g3','g4')}

membersL_noweight <- getMCI(cluster,test)
```

`getNetwork` *Building Networks of Nodes*

Description

This function builds one correlation network for each state (sample group) and runs across all states. The network nodes are defined by the context of the input dataset. For transcriptomic network analysis, network nodes can be the expressed transcript IDs and network links can be the correlation coefficients. Using the Pearson Correlation Coefficiency (PCC) analysis, this function assembles a correlation network of nodes (e.g., co-expressed transcripts) for each state using the R package igraph.

Usage

```
getNetwork(optimal, fdr = 0.05)
```

Arguments

`optimal`	A list of x numeric data frames, where x is the number of states studied. Each data frame consists of loci with high standard deviations. This object can be obtained through `sd_selection` function.
`fdr`	A numeric cutoff value for a Pearson Correlation Coefficiency (PCC) analysis. Default is 0.05. Transcripts are linked into a network if their correlations meet this PCC-significance criterion.

Value

A list of igraph objects whose length is the length of the input object `optimal`. Each object is a network of correlated nodes whose PCCs meet the significant criteria based on the false discovery rate (FDR) control. The length of the list is the number of states with PCC networks. If no PCC meets the significant criteria in a state, the state will be deleted from the output.

Author(s)

Zhezhen Wang <zhezhen@uchicago.edu>

Examples

```
test = list('state1' = matrix(sample(1:10,6),2,3),
 'state2'=matrix(sample(1:10,6),2,3),
 'state3' = matrix(sample(1:10,6),2,3))

for(i in names(test)){
  colnames(test[[i]]) = 1:3
  row.names(test[[i]]) = 1:2}

igraphL <- getNetwork(test, fdr=1)
#[1] "state1:2 nodes"
#[1] "state2:2 nodes"
#[1] "state3:2 nodes
```

`getReadthrough` *Overlapping Coding Regions*

Description

The `getReadthrough()` function is used to find long transcripts that cover more than two coding regions for gene regions of interst.

Usage

```
getReadthrough(gr, cod_gr)
```

Arguments

`gr`	A GRanges object that shows the start and end loci on genome.
`cod_gr`	A GRanges object contaning coding regions.

Details

For details of findOverlaps, type.partialOverlap, type.50Overlap type.toPlot, queryhits, readthrough and subjecthits see, GenomicRanges https://www.bioconductor.org/packages/release/bioc/html/GenomicRanges.html, IRanges https://www.bioconductor.org/packages/release/bioc/html/IRanges.html, and BiocManager http://bioconductor.org/install/index.html.

Value

A GRanges object that returns overlapping regions of the classified transcript biotypes.

Note

Replace the path_file when loading data locally to the data directory.

Author(s)

Zhezhen Wang and Biniam Feleke

Source

Reference GRCh37 genome https://www.gencodegenes.org/human/release_25lift37.html. For details on gtf format visit ensemble https://useast.ensembl.org/info/website/upload/gff.html.

References

Wang, Z. Z., J. M. Cunningham and X. H. Yang (2018).'CisPi: a transcriptomic score for disclosing cis-acting disease-associated lincRNAs.' Bioinformatics34(17): 664-670'

Examples

```
#First Load datasets and libraries
library(GenomicRanges)
data("gencode")
data("ILEF")
data("cod")

#Assigning datasets a GRanges object
gencode_gr = GRanges(gencode)
ILEF_gr = GRanges(ILEF)
cod_gr = GRanges(cod)

getReadthrough(ILEF_gr, cod_gr)

## Not run: getReadthrough(cod_gr)
```

`GSE6136_cli` *GSE6136 cli dataset*

Description

A gene annotation samples with their corresponding geneId extracted from GENCODE database. A python script was then run to extcat GSE6136_cli dataset. The script is included in the R/data_raw folder.

Usage

```
GSE6136_cli
```

Format

A dataframe with 22690 columns and 27 rows column.

GSM142398-GSM142423 Names of gene interest

Rows Summary of GSM genes

Source

https://www.gencodegenes.org/human/

`GSE6136_matrix` *GSE6136 matrix dataset*

Description

A gene annotation samples with their corresponding geneId extracted from GENCODE database. A python script was then run to extcat GSE6136_cli dataset. The script is included in the R/data_raw folder.

Usage

```
GSE6136_matrix
```

Format

A dataframe with 22690 columns and 27 rows column.

GSM142398-GSM142423 Names of gene interest

ID_REF Reference ID of the target genes

Source

https://www.gencodegenes.org/human/

`ILEF` *Chromosome ranges of chr21 dataset*

Description

A dataset containing chromosomes in the genome regions of interest for 137 chromosome. The variables are as follows:

Usage

```
ILEF
```

Format

A data frame of chr21 with 137 rows and 4 variables:

seqnames chromosome names (chr1,chr6,chr21)

start gene read start position (167684657,167729508)

end end of gene read position (15710335,43717938)

width width of gene

strand specific strand of the genomic location (+,-,*

Row.names name of the data rows(A1BG,vawser)

`intron` *Coding transcriptome in chr21 dataset*

Description

A dataset containing chromosomes in the genome regions of interest. The variables are as follows:

Usage

```
intron
```

Format

A data frame with 659327 rows and 5 variables:

seqnames chromosome names (chr1,chrM,chr21)

ranges chromosome ranges on the genome(167684657–167729508)

strand specific strand of the genomic location (+,-,*)

name internal assigned names(uc001aaa.3_intron_0_0_chr1_12228_f)

score score not used in this data set(0)

`optimize.sd_selection` *optimization of sd selection*

Description

The `optimize.sd_selection` filters a multi-state dataset based on a cutoff value for standard deviation per state and optimizes. By default, a cutoff value of 0.01 is used. Suggested if each state contains more than 10 samples.

Usage

```
optimize.sd_selection(df, samplesL, B = 100, percent = 0.8,
  times = 0.8, cutoff = 0.01, method = c("other", "reference",
  "previous", "itself", "longitudinal reference"), control_df = NULL,
  control_samplesL = NULL)
```

Arguments

`df` A dataframe of numerics. The rows and columns represent unique transcript IDs (geneID) and sample names, respectively.

`samplesL` A list of n vectors, where n equals to the number of states. Each vector gives the sample names in a state. Note that the vectors (sample names) has to be among the column names of the R object 'df'.

`B` An integer indicating number of times to run this optimization, default 1000.

`percent` A numeric value indicating the percentage of samples will be selected in each round of simulation.

`times` A numeric value indicating the percentage of `B` times a transcript need to be selected in order to be considered a stable signature.

`cutoff` A positive numeric value. Default is 0.01. If < 1, automatically goes to select top x# transcripts using the a selecting method (which is either the `reference`, `other` or `previous` stage), e.g. by default it will select top 1% of the transcripts.

`method` Selection of methods from `reference`, `other`, `previous`, default uses `other`. Partial match enabled.

- `itself`, or `longitudinal reference`. Some specific requirements for each option:
- `reference`, the reference has to be the first.
- `previous`, make sure sampleL is in the right order from benign to malign.
- `itself`, make sure the cutoff is smaller than 1.
- `longitudinal reference` make sure control_df and control_samplesL are not NULL. The row numbers of control_df is the same as df and all trancript in df is also in control_df.

`control_df` A count matrix with unique loci as row names and samples names of control samples as column names, only used for method `longitudinal reference`

`control_samplesL`
A list of characters with stages as names of control samples, required for method 'longitudinal reference'

Value

A list of dataframe of filtered transcripts with the highest standard deviation are selected from `df` based on a cutoff value assigned. The resulting dataframe represents a subset of the raw input `df`.

Author(s)

Zhezhen Wang <zhezhen@uchicago.edu>

See Also

sd_selection

Examples

```
counts = matrix(sample(1:100,30),2,30)
colnames(counts) = 1:30
row.names(counts) = paste0('loci',1:2)
cli = cbind(1:30,rep(c('state1','state2','state3'),each = 10))
colnames(cli) = c('samples','group')
samplesL <- split(cli[,1],f = cli[,'group'])
test_sd_selection <- optimize.sd_selection(counts, samplesL, B = 3, cutoff =0.01)
```

`plotBar_MCI` *plot MCI barplots*

Description

A barplot of MCI for all clusters in all states

Usage

```
plotBar_MCI(MCIl, ylim = NULL, nr = 1, nc = NULL, order = NULL,
  minsize = 3, states = NULL)
```

Arguments

`MCIl`	A list can be obtained through getMCI
`ylim`	A vector needed if the output barplots need to be on the same y scale
`nr`	The number of rows to plot
`nc`	The number of columns to plot, default length(groups)
`order`	A character vector of the order of the barplot. Default isNULL which uses the input list order
`minsize`	A non-negative numeric value of minimum size allowed for a cluster
`states`	State names should be shown on the plot. Default is NULL, assign this if you want to show all states including states without nodes.

Value

Return a barplot of MCI scores across states.

Author(s)

Zhezhen Wang <zhezhen@uchicago.edu>

Examples

```
test = list('state1' = matrix(sample(1:10,6),4,3),'state2' = matrix(sample(1:10,6),4,3),'state3' = matrix(sam
# assign colnames and rownames to the matrix
for(i in names(test)){
 colnames(test[[i]]) = 1:3
 row.names(test[[i]]) = c('g1','g2','g3','g4')
}

cluster = list(c(1,2,2,1),c(1,2,3,1),c(2,2,1,1))
names(cluster) = names(test)
for(i in names(cluster)){
 names(cluster[[i]]) = c('g1','g2','g3','g4')
}
membersL_noweight <- getMCI(cluster,test)
plotBar_MCI(membersL_noweight)
```

---

plotIc   *plot a line plot of Ic scores for each state.*

Description

plot a line plot with Ic score for each state

Usage

```
plotIc(Ic, las = 0, order = NULL)
```

Arguments

`Ic`	A vector with names of states. If order is not assigned, then plot by the order of this vector.
`las`	Numeric in 0,1,2,3; the style of axis labels. Default is 0, meaning labels are parallel. (link to http://127.0.0.1:21580/library/graphics/html/par.html)
`order`	A vector of state names in the customized order to be plotted, set to NULL by default.

Value

Return a line plot of Ic score across states

Author(s)

Zhezhen Wang <zhezhen@uchicago.edu>

Examples

```
Ic = c('state3' = 3.4,'state1' = 5.6,'state2' = 2)
plotIc(Ic,order = c('state1','state2','state3'))
```

`plotMaxMCI` *Plot the Maximized MCI per State*

Description

This function generates a line plot over multiple states with the maximum MCI score per state. The module size (i.e., number of nodes) is specified at each state in parentheses.

Usage

```
plotMaxMCI(maxMCIms, MCIl, las = 0, order = NULL, states = NULL)
```

Arguments

`maxMCIms`	A list of 2 elements. The 1st element is an integer vector of module ids whose names are the state names. The 2nd element is a list of character vectors per state. The vectors are network nodes (e.g. transcript ids). This parameter can be obtained by running function `getMaxMCImember`
`MCIl`	A list of numeric vectors whose names are unique cluster ids. Each vector represents the MCI scores of modules in a state. This can be the second element of the output from the function `getMCI`.
`las`	Numeric in 0,1,2,3; the style of axis labels. Default is 0, meaning labels are parallel. See `getMCI` for more detail
`order`	A vector of state names in the customized order to be plotted, set to NULL by default.
`states`	A character vector of state names that will be shown on the plot, set to NULL by default. Assign this if you want to show all states, including states with no resultind modules. This parameter will overwrite the parameter 'order'

Value

Returns a line plot of maximum MCI scores across the states

Author(s)

Zhezhen Wang <`zhezhen@uchicago.edu`>

Examples

```
maxMCIms = list(c(state1 = 1,state2 = 2,state3 = 1),c(list(state1 = c('g1','g2','g3'),state2 = c('g3','g5'),sta
MCIl = list(state1=c('1' = 8.84,'2' = 6.4),state2 = c('1' =NA,'2' = 9.5,'3' = NA),state3 = c('1' = 2.3,'2' = 1.4)
plotMaxMCI(maxMCIms,MCIl)
```

`plot_Ic_Simulation` *line plot of Ic score and simulated Ic scores*

Description

generate a line plot of Ic score and simulated Ic scores.

Usage

```
plot_Ic_Simulation(Ic, simulation, las = 0, ylim = NULL,
  order = NULL, main = NULL)
```

Arguments

`Ic`	A vector with names of states. If order is not assigned, then plot by the order of this vector.
`simulation`	A numeric matrix of Ic scores in which rows are states and columns are numbers of simulated times. It can be obtained from `simulation_Ic`
`las`	Numeric in 0,1,2,3; the style of axis labels. Default is 0, meaning labels are parallel. (link to http://127.0.0.1:21580/library/graphics/html/par.html)
`ylim`	An integer vector of length 2. Default is NULL.
`order`	A vector of state names in the customized order to be plotted, set to NULL by default.
`main`	A character vector. The title of the plot. Defualt is NULL.

Value

Return a line plot of Ic(red) and simulated Ic(grey) scores across all states

Author(s)

Zhezhen Wang <`zhezhen@uchicago.edu`>

Examples

```
sim = matrix(sample(1:10,9),3,3)
row.names(sim) = paste0('state',1:3)
Ic = c('state1' = 3.4,'state2' = 5.6,'state3' = 2)
plot_Ic_Simulation(Ic,sim)
```

`plot_MCI_Simulation` *Simulation of Loci to Calculate the CI Score*

Description

simulation of loci to calculate the Ic score.

Usage

```
plot_MCI_Simulation(MCI, simulation, las = 0, order = NULL,
  ylim = NULL, main = NULL)
```

Arguments

`MCI`	A named vector of max CI scores per state, can be obtained from function `getMaxStats`
`simulation`	A matrix state * number of simulated times, can be obtained from function `simulationMCI`
`las`	Numeric in 0,1,2,3; the style of axis labels. Default is 0, meaning labels are parallel. (link to http://127.0.0.1:21580/library/graphics/html/par.html)
`order`	A vector of state names in the customized order to be plotted, set to NULL by default.
`ylim`	An integer vector of length 2. Default is NULL.
`main`	A character vector. The title of the plot. Defualt is NULL.

Value

Return a line plot of MCI(red) and simulated MCI(grey) scores across all states

Author(s)

Zhezhen Wang <`zhezhen@uchicago.edu`>

Examples

```
MCI = c(1:3); names(MCI) = c('a','b','c')
simMCI = matrix(sample(1:100,9),3,3)
row.names(simMCI) = names(MCI)
plot_MCI_Simulation(MCI,simMCI)
```

plot_simulation_sample *calculating Ic scores based on simulated samples*

Description

simulate x samples B times to calculate the Ic score, where x should be the same as the length of identified BioTiP and B is self-defined.

Usage

```
plot_simulation_sample(counts, sampleNo, Ic, genes, B = 1000,
  main = "simulation of samples")
```

Arguments

counts	A numeric matrix or data frame. The rows and columns represent unique transcript IDs (geneID) and sample names, respectively.
sampleNo	An integer of sample size of the tipping point
Ic	A numeric value. Ic score of identified BioTiP
genes	A character vector of identified BioTiP unique ids
B	An integer indicating number of times to run this simulation, default 1000.
main	A character vector. The title of the plot. Defualt is NULL.

Value

Return a density plot of simulated Ic score with p-value

Author(s)

Zhezhen Wang <zhezhen@uchicago.edu>

Examples

```
counts = matrix(sample(1:100,27),3,9)
colnames(counts) = 1:9
row.names(counts) = c('loci1','loci2','loci3')
BioTiP = c('loci1','loci2')
plot_simulation_sample(counts,3,3.4,BioTiP,B=3)
```

`sd_selection` *Selecting Highly Oscillating Transcripts*

Description

`sd_selection` pre-selects highly oscillating transcripts from the input dataset `df`. The dataset must contain multiple sample groups (or 'states'). For each state, the function filters the dataset using a cutoff value for standard deviation. The default cutoff value is 0.01 (i.e., higher than the top 1% standard deviation).

Usage

```
sd_selection(df, samplesL, cutoff = 0.01, method = c("other",
  "reference", "previous", "itself", "longitudinal reference"),
  control_df = NULL, control_samplesL = NULL)
```

Arguments

`df`	A numeric matrix or data frame. The rows and columns represent unique transcript IDs (geneID) and sample names, respectively.
`samplesL`	A list of vectors, whose length is the number of states. Each vector gives the sample names in a state. Note that the vectors (sample names) has to be among the column names of the R object 'df'.
`cutoff`	A positive numeric value. Default is 0.01. If < 1, automatically selects top x transcripts using the a selecting method (which is either the `reference`, `other` stages or `previous` stage), e.g. by default it will select top 1% of the transcripts.
`method`	Selection of methods from `reference`, `other`, `previous`, default uses `other`. Partial match enabled. • `itself`, or `longitudinal reference`. Some specific requirements for each option: • `reference`, the reference has to be the first. • `previous`, make sure sampleL is in the right order from benign to malign. • `itself`, make sure the cutoff is smaller than 1. • `longitudinal reference` make sure control_df and control_samplesL are not NULL. The row numbers of control_df is the same as df and all trancript in df is also in control_df.
`control_df`	A count matrix with unique loci as row names and samples names of control samples as column names, only used for method `longitudinal reference`
`control_samplesL`	A list of characters with stages as names of control samples, required for method 'longitudinal reference'

Value

`sd_selection()` A list of data frames, whose length is the number of states. The rows in each data frame are the filtered transcripts with highest standard deviation selected from `df` and based on an assigned cutoff value. Each resulting data frame represents a subset of the raw input `df`, with the sample ID of the same state in the column.

Author(s)

Zhezhen Wang <zhezhen@uchicago.edu>

See Also

optimize.sd_selection

Examples

```
counts = matrix(sample(1:100,18),2,9)
colnames(counts) = 1:9
row.names(counts) = c('loci1','loci2')
cli = cbind(1:9,rep(c('state1','state2','state3'),each = 3))
colnames(cli) = c('samples','group')
samplesL <- split(cli[,1],f = cli[,'group'])
test_sd_selection <- sd_selection(counts, samplesL, 0.01)
```

---

simulationMCI *Get MCI Scores for Feature Permutation*

Description

This function gets the MCI scores for randomly selected features (e.g. transcript ids)

Usage

```
simulationMCI(len, samplesL, df, adjust.size = FALSE, B = 1000)
```

Arguments

len	A integer that is the length of BioTiP.
samplesL	A list of vectors, whose length is the number of states. Each vector gives the sample names in a state. Note that the vector s (sample names) has to be among the column names of the R object 'df'.
df	A numeric matrix or dataframe of numerics, factor or character. The rows and columns represent unique transcript IDs (geneID) and sample names, respectively
adjust.size	A boolean value indicating if MCI score should be adjust by module size (the number of transcripts in the module) or not. Default FALSE.
B	A integer, setting the permutation with B runs. Default is 1000.

Value

A numeric matrix indicating the MCI scores of permutation. The dimemsion (row * column) of this matrix is the length of `samplesL` * `B`.

Author(s)

Zhezhen Wang <zhezhen@uchicago.edu>

Examples

```
counts = matrix(sample(1:100,18),3,9)
colnames(counts) = 1:9
row.names(counts) = c('loci1','loci2','loci3')
cli = cbind(1:9,rep(c('state1','state2','state3'),each = 3))
colnames(cli) = c('samples','group')
samplesL <- split(cli[,1],f = cli[,'group'])
simMCI = simulationMCI(2,samplesL,counts,B=2)
```

`simulation_Ic` *calculating Ic scores based on simulated loci*

Description

simulate `x` loci `B` to calculate the Ic score, where `x` should be the same as the length of identified BioTiP and `B` is self-defined.

Usage

```
simulation_Ic(obs.x, sampleL, counts, B = 1000)
```

Arguments

`obs.x`	A integer, length of identified BioTiP
`sampleL`	A list of vectors, whose length is the number of states. Each vector gives the sample names in a state. Note that the vector s (sample names) has to be among the column names of the R object 'df'.
`counts`	A numeric matrix or dataframe in which columns are samples and rows are transcripts. Each row needs to have a unique row name (i.e. transcript ID)
`B`	A integer, setting the permutation with `B` runs. Default is 1000.

Value

A matrix of `y` rows and `B` columns where `y` is the length of sampleL and `B` is self-defined. Each column is a set of Ic scores calculated for each state

Author(s)

Zhezhen Wang <`zhezhen@uchicago.edu`>

Examples

```
counts = matrix(sample(1:100,27),3,9)
colnames(counts) = 1:9
row.names(counts) = c('loci1','loci2','loci3')
cli = cbind(1:9,rep(c('state1','state2','state3'),each = 3))
colnames(cli) = c('samples','group')
samplesL <- split(cli[,1],f = cli[,'group'])
simulation_Ic(2,samplesL,counts,B =3)
```

Index