Package ‘RegEnrich’

November 17, 2025

Type Package

Title Gene regulator enrichment analysis

Version 1.21.0

Description This package is a pipeline to identify the key gene regulators in a biological process, for example in cell differentiation and in cell development after stimulation. There are four major steps in this pipeline: (1) differential expression analysis; (2) regulator-target network inference; (3) enrichment analysis; and (4) regulators scoring and ranking.

Depends R (>= 4.0.0), S4Vectors, dplyr, tibble, BiocSet, SummarizedExperiment

License GPL (>= 2)

Encoding UTF-8

VignetteBuilder knitr

RoxygenNote 7.3.1

Collate 'COEN.R' 'globals.R' 'DEA.R' 'GRN.R' 'data.R' 'regenrichClasses.R' 'genericMethods.R' 'localUtils.R' 'plots.R' 'regFET.R' 'regSEA.R' 'regenrich_diffExpr.R' 'regenrich_enrich.R' 'regenrich_network.R' 'regenrich_rankScore.R' 'results.R' 'show.R' 'topNet.R'

Imports randomForest, fgsea, DOSE, BiocParallel, DESeq2, limma, WGCNA, ggplot2 (>= 2.2.0), methods, reshape2, magrittr, BiocStyle

Suggests GEOquery, rmarkdown, knitr, BiocManager, testthat

biocViews GeneExpression, Transcriptomics, RNASeq, TwoChannel, Transcription, GeneTarget, NetworkEnrichment, DifferentialExpression, Network, NetworkInference, GeneSetEnrichment, FunctionalPrediction

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

git_branch devel

git_last_commit 73e2009

git_last_commit_date 2025-10-29

Repository Bioconductor 3.23

Date/Publication 2025-11-16

Author Weiyang Tao [cre, aut],
Aridaman Pandit [aut]

Maintainer Weiyang Tao <weiyangtao1513@gmail.com>

Contents

DeaSet-class . . . 2
dim,TopNetwork-method . . . 3
Enrich-class . . . 4
getResultsNames . . . 5
head,Score-method . . . 6
Lyme_GSE63085 . . . 7
newDeaSet . . . 7
newTopNetwork . . . 8
plotOrders . . . 9
plotRegTarExpr . . . 10
plotSoftPower . . . 12
plot_Enrich . . . 13
print.Score . . . 14
reexports . . . 15
RegenrichSet . . . 16
RegenrichSet-class . . . 23
regenrich_diffExpr . . . 24
regenrich_enrich . . . 25
regenrich_network . . . 27
regenrich_rankScore . . . 29
results_expr . . . 30
Score-class . . . 32
show,DeaSet-method . . . 33
TFs . . . 34
TopNetwork-class . . . 34

Index **35**

`DeaSet-class` *DeaSet class*

Description

DeaSet class

Slots

`colData` DataFrame object, sample information, the row name is corresponding to the column names of expression matrix in the `assays` slot.

`assays` SimpleList object of one/multiple matrix/matrices, this is the slot for storing the expression data after filtering (and after Variance Stabilizing Transformation, i.e. VST, if the differential analysis method is 'Wald_DESeq2' or 'LRT_DESeq2'). And the expression matrix is used for network inference and plotting.

`NAMES` row names of expression data in `assays` slot and `elementMetadata` slot.

`elementMetadata` feature information, contains at least a DataFrame of three columns, i.e. ‘gene‘, ‘p‘ and ‘logFC‘, which stores gene names/IDs, differential p values and log2 expression fold changes, respectively.

`metadata` DataFrame object, information of feature columns.

`assayRaw` a slot for saving the raw expression data.

Examples

```
nrows = 100
ncols = 6
counts = matrix(rnbinom(nrows * ncols, size = 2, mu = 500),
                nrow = nrows)
assays = SimpleList(assayData = counts)

colData = DataFrame(Condition = rep(c("treatment", "ctrl"), 3),
                    row.names=LETTERS[1:6])
geneNames = sprintf("G%03s", seq(nrows))
elementMetadata = DataFrame(gene = geneNames,
                            p = numeric(nrows),
                            logFC = numeric(nrows))

ds = new("DeaSet",
         assays = Assays(assays),
         colData = colData,
         assayRaw = counts,
         elementMetadata = elementMetadata,
         NAMES = geneNames)
ds
```

`dim,TopNetwork-method` *dimention of ‘TopNetwork‘ object*

Description

dimention of ‘TopNetwork‘ object

Usage

```
## S4 method for signature 'TopNetwork'
dim(x)
```

Arguments

x	a ‘TopNetwork‘ object.

Value

Dimention of regulator-target network edge table.

Examples

```
nw = newTopNetwork()
dim(nw)
```

Enrich-class *Enrich class*

Description

The ‘Enrich‘ object is to store enrichment analysis results by either ‘FET‘ method or ‘GSEA‘ method.

Slots

`topResult` data frame. The enrichment results that pass thresholds (default threshold is 0.05).

`allResult` data frame. The enrichment results by FET or GSEA methods.

`gene` character vector indicating the genes used for enrichment analysis.

`namedScores` numeric vector, a vector of ranked scores (decendent), the names of the scores are the genes to perform enrichment analysis. Here the scores are p-value of each gene.

`type` character indicating enrichment method, either ’FET’ or ’GSEA’.

getResultsNames *Inference the name of results of DESeq analysis by a formula (or model matrix) and sample information*

Description

Inference the name of results of DESeq analysis by a formula (or model matrix) and sample information

Usage

```
getResultsNames(design, pData = NULL)
```

Arguments

design	either a formula or a model matrix.
pData	a data frame, showing the information of each sample. If design is a formula, the pData must be include the columns that identical to the terms of the design formula. If design is a model matrix, then pData is not used. Default is NULL.

Value

the names of contrast parameter (list of character format) that regenrich_diffExpr and results function can use, and it is the same as the value that resultsNames function returns.

Examples

```
# formula with intercept
design = ~condition
pData = data.frame(condition = factor(c('A', 'A', 'A', 'B', 'B', 'B'),
                                      c('A', 'B')))
getResultsNames(design, pData)

# formula without intercept
design = ~0+condition
getResultsNames(design, pData)

# formula with two terms
design = ~condition+treatment
pData = data.frame(condition = factor(rep(c('A', 'B'), each= 4),
                                      c('A', 'B')),
                   treatment = factor(rep_len(c('Ctrl', 'Treat'), 8),
                                      c('Ctrl', 'Treat')))
getResultsNames(design, pData)

# formula with two terms and an interaction term
design = ~condition+treatment+condition:treatment
getResultsNames(design, pData)
```

```
# design is a model matrix
pData = data.frame(condition = factor(rep(c('A', 'B'), each= 4),
                                      c('A', 'B')),
                   treatment = factor(rep_len(c('Ctrl', 'Treat'), 8),
                                      c('Ctrl', 'Treat')))
design = model.matrix(~condition+treatment, pData)
getResultsNames(design)
```

`head,Score-method` *head or tail of Score object*

Description

head or tail of Score object

Usage

```
## S4 method for signature 'Score'
head(x, ...)

## S4 method for signature 'Score'
tail(x, ...)
```

Arguments

`x`	an `Score` object.
`...`	arguments to be passed to or from other methods.

Value

Head or tail table of Score object.

Examples

```
s = newScore(letters, seq(26), seq(26), seq(26), seq(2, 0, len = 26))
s1 = head(s)
s1

s2 = tail(s)
s2
```

Lyme_GSE63085 *Example RNAseq dataset [Human]*

Description

Data from an RNA sequencing experiment on peripheral mononuclear blood cells (PBMC) of Lyme disease patients against healthy controls. It contains a gene expression (FPKM) table (data frame) and a sample information table (data frame).

Usage

```
data(Lyme_GSE63085)
```

Format

A list of 2 elements: FPKM and sampleInfo. FPKM is the 'Fragments Per Kilobase of transcript per Million mapped reads' data, which is a 5000 (genes) * 52 (samples) data frame. sampleInfo is the information of samples, which is 52 (samples) * 9 (features) data frame. The full version of FPKM table contains 23615 rows, which can be downloaded from GEO database.

Source

URL

References

Bouquet et al. (2016) mBio 7(1): e00100-16 (PubMed)

newDeaSet *DeaSet object creator*

Description

DeaSet object creator

Usage

```
newDeaSet(
  assayRaw = matrix(nrow = 0, ncol = 0),
  rowData = NULL,
  assays = SimpleList(),
  colData = DataFrame(),
  metadata = list()
)
```

Arguments

assayRaw	A matrix of gene expression data. This can be the same as the matrix-like element in assays parameter.
rowData	A DataFrame object describing the rows.
assays	A list or SimpleList of matrix-like element, or a matrix-like object. The matrix-like element can be the same as assayRaw parameter.
colData	A DataFrame describing the sample information.
metadata	An optional list of arbitrary content describing the overall experiment.

Value

A DeaSet object.

Examples

```
# Empty DeaSet object
newDeaSet()

# 100 * 6 DeaSet object
nrows = 100
ncols = 6
counts = matrix(rnbinom(nrows * ncols, size = 2, mu = 500),
                nrow = nrows)
assays = SimpleList(counts=counts)

colData = DataFrame(Condition = rep(c("treatment", "ctrl"), 3),
                    row.names=LETTERS[1:6])
geneNames = sprintf("G%03s", seq(nrows))
elementMetadata = DataFrame(gene = geneNames,
                            p = numeric(nrows),
                            logFC = numeric(nrows))
newDeaSet(assayRaw = counts,
          rowData = elementMetadata,
          assays = SimpleList(assayData = counts),
          colData = colData)
```

---

newTopNetwork *TopNetwork object creator*

Description

This function create ‘TopNetwork‘ object using 3-column edge table.

Usage

```
newTopNetwork(
  networkEdgeTable,
  reg = "",
  directed = TRUE,
  networkConstruction = c("new", "COEN", "GRN"),
  percent = 100
)
```

Arguments

`networkEdgeTable`
: a data frame of 3 columns, representing 'from.gene' ('regulators'), 'to.gene' ('targets') and 'weight', respectively.

`reg`
: a vector of gene regulators.

`directed`
: logical, whether the network is directed. Default is TRUE.

`networkConstruction`
: the method to construct this network. Possible can be: 'COEN', coexpression network; 'GRN', gene regulatory network by random forest; 'new' (default), meaning a network provided by user, rather than infered based on the expression data.

`percent`
: the percentage of edges in the original whole network. Default is 100, meaning 100% edges in whole network.

Value

an object of topNetwork class.

Examples

```
data(TFs)
edge = data.frame(from = rep(TFs$TF_name[seq(3)], seq(3)),
                  to = TFs$TF_name[11:16], weight = 0.1*(6:1))
object = newTopNetwork(edge, networkConstruction = 'new', percent = 100)
object
str(object)
```

---

plotOrders *Compare the orders of two vectors*

Description

compare the orders of two vectors

Usage

```
plotOrders(name1, name2)
```

Arguments

name1	a vector with first order.
name2	a vector with anothoer second order.

Value

A plot of comparing two orders of vectors.

Examples

```
a = c('a1', 'a2', 'a5', 'a4')
b = c( 'a2', 'a5', 'a7', 'a4', 'a6')
plotOrders(a, b)
```

---

plotRegTarExpr *Plot regulator and its targets expression*

Description

Plot regulator and its targets expression

Usage

```
plotRegTarExpr(
  object,
  reg,
  n = 1000,
  scale = TRUE,
  tarCol = "black",
  tarColAlpha = 0.1,
  regCol = "#ffaa00",
  xlab = "Samples",
  ylab = "Z-scores",
  ...
)
```

Arguments

object	a RegenrichSet object, to which at least regenrich_diffExpr and regenrich_network functions have been applied.
reg	a regulator to plot.
n	the maximun number of targets to plot.
scale	logical, whether gene expression is z-score normalized.
tarCol	the color of the lines for the targets of the regulator.
tarColAlpha	numeric, ranging from 0 to 1, indicating transparancy of target lines.

regCol	the color of the line for the 'reg'.
xlab	x label of plot.
ylab	y label of plot.
...	other parameters in ggplot function.

Value

a ggplot object.

Examples

```
# constructing a RegenrichSet object
colData = data.frame(patientID = paste0('Sample_', seq(50)),
                     week = rep(c('0', '1'), each = 25),
                     row.names = paste0('Sample_', seq(50)),
                     stringsAsFactors = TRUE)
design = ~week
reduced = ~1
set.seed(123)
cnts = matrix(as.integer(rnbinom(n=1000*50, mu=100, size=1/0.1)), ncol=50,
              dimnames = list(paste0('gene', seq(1000)), rownames(colData)))

cnts[5,26:50] = cnts[5,26:50] + 50L # add reads to gene5 in some samples.
id = sample(31:1000, 20) # randomly select 20 rows, and assign reads.
cnts[id,] = vapply(cnts[5,], function(x){
  as.integer(rnbinom(n = 20, size = 1/0.02, mu = x))},
  FUN.VALUE = rep(1L, 20))

object = RegenrichSet(expr = cnts,
                      colData = colData,
                      method = 'LRT_DESeq2', minMeanExpr = 0,
                      design = design, reduced = reduced, fitType = 'local',
                      networkConstruction = 'COEN',
                      enrichTest = 'FET',
                      reg = paste0('gene', seq(30)))

## RegEnrich analysis
object = regenrich_diffExpr(object)

# Set a random softPower, otherwise it is difficult to achive a
# scale-free network because of a randomly generated count data.
object = regenrich_network(object, softPower = 3)
object = regenrich_enrich(object)
object = regenrich_rankScore(object)

## plot expression of a regulator and its targets.
plotRegTarExpr(object, reg = 'gene5')
plotRegTarExpr(object, reg = 'gene27')
```

plotSoftPower *Plot soft power for WGCNA analysis*

Description

Plot soft power and corresponding scale free topology fitting index to find a proper soft power for WGCNA analysis.

Usage

```
plotSoftPower(
  expr,
  rowSample = FALSE,
  weights = NULL,
  powerVector = c(seq(10), seq(12, 20, by = 2)),
  RsquaredCut = 0.85,
  networkType = "unsigned",
  removeFirst = FALSE,
  nBreaks = 10,
  corFnc = WGCNA::cor,
  corOptions = list(use = "p")
)
```

Arguments

expr	Gene expression data, either a matrix or a data frame. By default, each row represents a gene, each column represents a sample.
rowSample	logic. If TRUE, each row represents a sample. The default is FALSE.
weights	optional observation weights for expr to be used in correlation calculation.
powerVector	a vector of soft thresholding powers for which the scale free topology fit indices are to be calculated.
RsquaredCut	desired minimum scale free topology fitting index R^2. The default is 0.85.
networkType	character, network type. Allowed values are (unique abbreviations of) "unsigned" (default), "signed", "signed hybrid". See adjacency.
removeFirst	should the first bin be removed from the connectivity histogram? The default is FALSE.
nBreaks	number of bins in connectivity histograms. The default is 10.
corFnc	correlation function to be used in adjacency calculation. The default is the cor function in WGCNA.
corOptions	a named list of options to the correlation function specified in corFnc. The default is list(use = "p").

Value

a list of three elements: `powerEstimate`, `fitIndices`, and `plot`. `powerEstimate` is an estimate of an appropriate soft-thresholding power. `fitIndices` is a data frame containing the fit indices for scale free topology. The `plot` is a ggplot object.

Examples

```
data(Lyme_GSE63085)
log2FPKM = log2(Lyme_GSE63085$FPKM + 1)
log2FPKMhi = log2FPKM[rowMeans(log2FPKM) >= 10^-3, , drop = FALSE]
log2FPKMhi = head(log2FPKMhi, 3000) # First 3000 genes for example

softP = plotSoftPower(log2FPKMhi, RsquaredCut = 0.85)
print(softP)
```

---

plot_Enrich *Plot results of FET/GSEA enrichment analysis*

Description

Plot FET/GSEA enrichment results. If the FET method is applied, the top ‘showCategory‘ regulator will be plotted. If the GSEA method is applied, the GSEA graph of regulator ‘reg‘ will be plotted.

Usage

```
plot_Enrich(object, ...)

## S4 method for signature 'RegenrichSet'
plot_Enrich(
  object,
  reg = NULL,
  showCategory = 20,
  regDescription = NULL,
  font.size = 12
)
```

Arguments

`object`	a `RegenrichSet` object.
`...`	other parameters.
`reg`	The regulator to plot. This only works when the GSEA enrichment method has used.
`showCategory`	the number of regulator to plot.

regDescription	NULL or a two-column data frame, in which first column is the regulator IDs (for example ENSEMBL IDs), and the second column is the description of regulators (for example gene name). Default is NULL, meaning both columns are the same regulator names/IDs in the network.
font.size	font size of axis labels and axis tick mark labels, default is 12.

Value

a ggplot object of plotting FET or GSEA enrichment result.

Examples

```
# library(RegEnrich)
data("Lyme_GSE63085")
data("TFs")

data = log2(Lyme_GSE63085$FPKM + 1)
colData = Lyme_GSE63085$sampleInfo

# Take first 2000 rows for example
data1 = data[seq(2000), ]

design = model.matrix(~0 + patientID + week, data = colData)
object = RegenrichSet(expr = data1,
                      colData = colData,
                      method = "limma", minMeanExpr = 0,
                      design = design,
                      contrast = c(rep(0, ncol(design) - 1), 1),
                      networkConstruction = "COEN",
                      enrichTest = "FET")
# Differential expression analysis
object = regenrich_diffExpr(object)
# Network inference using "COEN" method
object = regenrich_network(object)
# Enrichment analysis by Fisher's exact test (FET)
object = regenrich_enrich(object)
# plot
plot_Enrich(object)

# Enrichment analysis by Fisher's exact test (FET)
object = regenrich_enrich(object, enrichTest = "GSEA")
# plot
plot_Enrich(object)
```

---

print.Score *Print Score object*

Description

Print Score object

Usage

```
## S3 method for class 'Score'
print(x, ...)
```

Arguments

x	a Score object.
...	optional arguments to print.

Value

print.Score returns the a Score object

Examples

```
x = newScore(letters[1:5], 1:5, 1:5, -2:2, seq(2, 1, len = 5))
print(x)
```

---

reexports *Objects exported from other packages*

Description

These objects are imported from other packages. Follow the links below to see their documentation.

magrittr %>%

Examples

```
# library(RegEnrich)
data("Lyme_GSE63085")
data("TFs")

data = log2(Lyme_GSE63085$FPKM + 1)
colData = Lyme_GSE63085$sampleInfo
data1 = data[seq(2000), ]

design = model.matrix(~0 + patientID + week, data = colData)

# Initializing a 'RegenrichSet' object
object = RegenrichSet(expr = data1,
                      colData = colData,
                      method = 'limma', minMeanExpr = 0,
                      design = design,
                      contrast = c(rep(0, ncol(design) - 1), 1),
                      networkConstruction = 'COEN',
                      enrichTest = 'FET')

# Using %>%
```

```
object %>% regenrich_diffExpr()
```

RegenrichSet *RegenrichSet object creator*

Description

This is ‘RegenrichSet‘ object creator function. There are four types of parameters in this function.
First, parameters to provide raw data and sample information;
‘expr‘ and ‘colData‘.

Second, parameters to perform differential expression analysis;
‘method‘, ‘minMeanExpr‘, ‘design‘, ‘reduced‘, ‘contrast‘, ‘coef‘, ‘name‘, ‘fitType‘, ‘sfType‘, ‘betaPrior‘, ‘minReplicatesForReplace‘, ‘useT‘, ‘minmu‘, ‘parallel‘, ‘BPPARAM‘ (also for network inference), ‘altHypothesis‘, ‘listValues‘, ‘cooksCutoff‘, ‘independentFiltering‘, ‘alpha‘, ‘filter‘, ‘theta‘, ‘filterFun‘, ‘addMLE‘, ‘blind‘, ‘ndups‘, ‘spacing‘, ‘block‘, ‘correlation‘, ‘weights‘, ‘proportion‘, ‘stdev.coef.lim‘, ‘trend‘, ‘robust‘, and ‘winsor.tail.p‘.

Thrid, parameters to perform regulator-target network inference;
‘reg‘, ‘networkConstruction‘, ‘topNetPercent‘, ‘directed‘, ‘rowSample‘, ‘softPower‘, ‘networkType‘, ‘TOMDenom‘, ‘RsquaredCut‘, ‘edgeThreshold‘, ‘K‘, ‘nbTrees‘, ‘importanceMeasure‘, ‘trace‘, ‘BPPARAM‘ (also for differential expression analysis), and ‘minR‘.

Fourth, parameters to perform enrichment analysis:
‘enrichTest‘, ‘namedScoresCutoffs‘, ‘minSize‘, ‘maxSize‘, ‘pvalueCutoff‘, ‘qvalueCutoff‘, ‘regAltName‘, ‘universe‘, and ‘nperm‘.

Usage

```
RegenrichSet(
  expr,
  colData,
  rowData = NULL,
  method = c("Wald_DESeq2", "LRT_DESeq2", "limma", "LRT_LM"),
  minMeanExpr = NULL,
  design,
  reduced,
  contrast,
  coef = NULL,
  name,
  fitType = c("parametric", "local", "mean"),
  sfType = c("ratio", "poscounts", "iterate"),
  betaPrior,
```

```
minReplicatesForReplace = 7,
useT = FALSE,
minmu = 0.5,
parallel = FALSE,
BPPARAM = bpparam(),
altHypothesis = c("greaterAbs", "lessAbs", "greater", "less"),
listValues = c(1, -1),
cooksCutoff,
independentFiltering = TRUE,
alpha = 0.1,
filter,
theta,
filterFun,
addMLE = FALSE,
blind = FALSE,
ndups = 1,
spacing = 1,
block = NULL,
correlation,
weights = NULL,
proportion = 0.01,
stdev.coef.lim = c(0.1, 4),
trend = FALSE,
robust = FALSE,
winsor.tail.p = c(0.05, 0.1),
reg = TFs$TF_name,
networkConstruction = c("COEN", "GRN", "new"),
topNetPercent = 5,
directed = FALSE,
rowSample = FALSE,
softPower = NULL,
networkType = "unsigned",
TOMDenom = "min",
RsquaredCut = 0.85,
edgeThreshold = NULL,
K = "sqrt",
nbTrees = 1000,
importanceMeasure = "IncNodePurity",
trace = FALSE,
minR = 0.3,
enrichTest = c("FET", "GSEA"),
namedScoresCutoffs = 0.05,
minSize = 5,
maxSize = 5000,
pvalueCutoff = 0.05,
qvalueCutoff = 0.2,
regAltName = NULL,
universe = NULL,
```

```
  nperm = 10000
)
```

Arguments

`expr` matrix or data.frame, expression profile of a set of genes or a set of proteins. If the `method = 'Wald_DESeq2' or 'LRT_DESeq2'` only non-negative integer matrix (read counts by RNA sequencing) is accepted.

`colData` data frame, sample phenotype data. The rows of colData must correspond to the columns of expr.

`rowData` NULL or data frame, information of each row/gene. Default is NULL, which will generate a DataFrame of three columns, i.e., "gene", "p", and "logFC".

`method` either 'Wald_DESeq2', 'LRT_DESeq2', 'limma', or 'LRT_LM' for the differential expression analysis.

- When method = 'Wald_DESeq2', the Wald test in DESeq2 package is used;
- When method = 'LRT_DESeq2', the likelihood ratio test (LRT) in DESeq2 package is used;
- When method = 'limma', the 'ls' method and empirical Bayes method in limma package are used to calculate moderated t-statistics and differential p-values;
- When method = 'LRT_LM', a likelihood ratio test is performed for each row of 'expr' to compare two linear model specified by 'design' and 'reduced' arguments. In this case, the fold changes are not calculated but set to 0.

`minMeanExpr` numeric, the cutoff of gene average expression for pre-filtering. The rows of 'expr' with everage expression < minMeanExpr is removed. The higher 'minMeanExpr' is, the more genes are not included for testing.

`design` either model formula or model matrix. For method = 'LRT_DESeq2' or 'LRT_LM', the design is the full model formula/matrix. For method = 'limma', and if design is a formula, the model matrix is constructed using model.matrix(design, colData), so the name of each term in the design formula must be included in the column names of 'colData'.

`reduced` The argument is used only when method = 'LRT_DESeq2' or 'LRT_LM', it is a reduced formula/matrix to compare against. If the design is a model matrix, 'reduced' must also be a model matrix.

`contrast` The argument is used only when method = 'LRT_DESeq2', 'Wald_DESeq2', or 'limma'.
When method = 'LRT_DESeq2', or 'Wald_DESeq2', it specifies what comparison to extract from the 'DESeqDataSet' object to build a results table (when method = 'LRT_DESeq2', this does not affect the value of 'stat', 'pvalue', or 'padj').
It can be one of following three formats:

- a character vector with exactly three elements: the name of a factor in the design formula, the name of the numerator level for the fold change, and the name of the denominator level for the fold change;

- a list of 1 or 2 character vector(s): the first element specifies the names of the fold changes for the numerator, and the second element (optional) specifies the names of the fold changes for the denominator. These names should be elements of `getResultsNames(design, colData)`;
- a numeric contrast vector with one element for each element in `getResultsNames(design, colData)`.

When method = 'limma', It can be one of following two formats:

- a numeric matrix with rows corresponding to coefficients in design matrix and columns containing contrasts;
- a numeric vector if there is only one contrast. Each element of the vector corresponds to coefficients in design matrix. This is similar to the third format of contrast when method = 'LRT_DESeq2', or 'Wald_DESeq2'.

`coef` The argument is used only when method = 'limma'. (Vector of) column number or column name specifying which coefficient or contrast of the linear model is of interest. Default is NULL.

`name` The argument is used only when method = 'LRT_DESeq2' or 'Wald_DESeq2'. the name of the individual effect (coefficient) for building a results table. Use this argument rather than contrast for continuous variables, individual effects or for individual interaction terms. The value provided to name must be an element of `getResultsNames(design, colData)`.

`fitType` either 'parametric', 'local', or 'mean' for the type of fitting of dispersions to the mean intensity. This argument is used only when method = 'Wald_DESeq2' or 'LRT_DESeq2'. See `DESeq` from DESeq2 package for more details. Default is 'parametric'.

`sfType` either 'ratio', 'poscounts', or 'iterate' for the type of size factor estimation. This argument is used only when method = either 'Wald_DESeq2' or 'LRT_DESeq2'. See `DESeq` from DESeq2 package for more details. Default is 'ratio'.

`betaPrior` This argument is used only when method = either 'Wald_DESeq2' or 'LRT_DESeq2'. See `DESeq` from DESeq2 package for more details.

`minReplicatesForReplace` This argument is used only when method = either 'Wald_DESeq2' or 'LRT_DESeq2'. See `DESeq` from DESeq2 package for more details. Default is 7.

`useT` This argument is used only when method = either 'Wald_DESeq2' or 'LRT_DESeq2'. See `DESeq` from DESeq2 package for more details. Default is FALSE,

`minmu` This argument is used only when method = either 'Wald_DESeq2' or 'LRT_DESeq2'. See `DESeq` from DESeq2 package for more details. Default is 0.5.

`parallel` whether computing (only for differential analysis with method = "Wald_DESeq2" or "LRT_DESeq2") is parallel (default is FALSE).

`BPPARAM` parameters for parallel computing (default is `bpparam()`).

`altHypothesis` = c('greaterAbs', 'lessAbs', 'greater', 'less'). This argument is used only when method = either 'Wald_DESeq2' or 'LRT_DESeq2'. See `results` from DESeq2 package for more details. Default is 'greaterAbs'.

`listValues` This argument is used only when method = either 'Wald_DESeq2' or 'LRT_DESeq2'. See `results` from DESeq2 package for more details. Default is c(1, -1),

`cooksCutoff` theshold on Cook's distance, such that if one or more samples for a row have a distance higher, the p-value for the row is set to NA. This argument is used only when method = either 'Wald_DESeq2' or 'LRT_DESeq2'. See `results` from DESeq2 package for more details.

`independentFiltering` logical, whether independent filtering should be applied automatically. This argument is used only when method = either 'Wald_DESeq2' or 'LRT_DESeq2'. See `results` from DESeq2 package for more details. Default is TRUE.

`alpha` the significance cutoff used for optimizing the independent filtering. This argument is used only when method = either 'Wald_DESeq2' or 'LRT_DESeq2'. See `results` from DESeq2 package for more details. Default is 0.1,

`filter` the vector of filter statistics over which the independent filtering is optimized. By default the mean of normalized counts is used. This argument is used only when method = either 'Wald_DESeq2' or 'LRT_DESeq2'. See `results` from DESeq2 package for more details.

`theta` the quantiles at which to assess the number of rejections from independent filtering. This argument is used only when method = either 'Wald_DESeq2' or 'LRT_DESeq2'. See `results` from DESeq2 package for more details.

`filterFun` an optional custom function for performing independent filtering and p-value adjustment. This argument is used only when method = either 'Wald_DESeq2' or 'LRT_DESeq2'. See `results` from DESeq2 package for more details.

`addMLE` if betaPrior=TRUE was used, whether the 'unshrunken' maximum likelihood estimates (MLE) of log2 fold change should be added as a column to the results table. This argument is used only when method = either 'Wald_DESeq2' or 'LRT_DESeq2'. See `results` from DESeq2 package for more details. Default is FALSE.

`blind` logical, whether to blind the transformation to the experimental design. This argument is used only when method = either 'Wald_DESeq2' or 'LRT_DESeq2'. See `vst` from DESeq2 package for more details. Default is FALSE, which is different from the default of vst function.

`ndups` positive integer giving the number of times each distinct probe is printed on each array. This argument is used only when method = 'limma'. See `lmFit` from limma package for more details. Default is 1.

`spacing` positive integer giving the spacing between duplicate occurrences of the same probe, spacing=1 for consecutive rows. This argument is used only when method = 'limma'. See `lmFit` from limma package for more details. Default is 1.

`block` vector or factor specifying a blocking variable on the arrays. Has length equal to the number of arrays. Must be NULL if ndups > 2. This argument is used only when method = 'limma'. See `lmFit` from limma package for more details. Default is NULL.

`correlation` the inter-duplicate or inter-technical replicate correlation. The correlation value should be estimated using the `duplicateCorrelation` function. This argument is used only when method = 'limma'. See `lmFit` from limma package for more details.

`weights` non-negative precision weights. Can be a numeric matrix of individual weights of same size as the object expression matrix, or a numeric vector of array weights with length equal to ncol of the expression matrix, or a numeric vector of gene weights with length equal to nrow of the expression matrix. This argument is used only when method = 'limma' or 'LRT_LM'. See `lmFit` from limma package for more details. Default is NULL.

`proportion` numeric value between 0 and 1, assumed proportion of genes which are differentially expressed. This argument is used only when method = 'limma'. See `eBayes` from limma package for more details. Default is 0.01.

`stdev.coef.lim` numeric vector of length 2, assumed lower and upper limits for the standard deviation of log2-fold-changes for differentially expressed genes. This argument is used only when method = 'limma'. See `eBayes` from limma package for more details. Default is c(0.1, 4).

`trend` logical, should an intensity-trend be allowed for the prior variance? This argument is used only when method = 'limma'. See `eBayes` from limma package for more details. Default is FALSE, meaning that the prior variance is constant.

`robust` logical, should the estimation of df.prior and var.prior be robustified against outlier sample variances? This argument is used only when method = 'limma'. See `eBayes` from limma package for more details. Default is FALSE.

`winsor.tail.p` numeric vector of length 1 or 2, giving left and right tail proportions of x to Winsorize. Used only when method = 'limma' and robust=TRUE. See `eBayes` from limma package for more details. Default is c(0.05,0.1)

`reg` a vector of regulator names (ID). By default, these are transcription (co-)factors defined by three literatures/databases, namely RegNet, TRRUST, and Marbach2016. The type (for example ENSEMBL gene ID, Entrez gene ID, or gene symble/name) of names or IDs of these regulators must be the same as the type of names or IDs in the regulator-target network.

`networkConstruction`
the method to construct this network. Possible can be:
'COEN', coexpression network;
'GRN', gene regulatory network by random forest;
'new' (default), meaning a network provided by user, rather than infered based on the expression data.

`topNetPercent` numeric, what percentage of the top edges in the full network is ratained. Default is 5, meaning top 5% of edges. This value must be between 0 and 100.

`directed` logical, whether the network is directed. Default is FALSE.

`rowSample` logic, if TRUE, each row represents a sample. Otherwise, each column represents a sample. Default is FALSE.

`softPower` numeric, a soft power to achieve scale free topology. If not provided, the parameter will be picked automatically by `plotSoftPower` function.

`networkType` network type. Allowed values are (unique abbreviations of) 'unsigned' (default), 'signed', 'signed hybrid'. See `adjacency`.

`TOMDenom` a character string specifying the TOM variant to be used. Recognized values are 'min' giving the standard TOM described in Zhang and Horvath (2005),

and 'mean' in which the min function in the denominator is replaced by mean. The 'mean' may produce better results but at this time should be considered experimental.

`RsquaredCut` desired minimum scale free topology fitting index R^2. Default is 0.85.

`edgeThreshold` numeric, the threshold to remove the low weighted edges, Default is NULL, which means no edges will be removed.

`K` integer or character. The number of features in each tree, can be either a integer number, 'sqrt', or 'all'. 'sqrt' denotes sqrt(the number of 'reg'), 'all' means the number of 'reg'. Default is 'sqrt'.

`nbTrees` integer. The number of trees. Default is 1000.

`importanceMeasure`
character. importanceMeasure can be '%IncMSE' or 'IncNodePurity', corresponding to type = 1 and 2 in `importance` function, respectively. Default is 'IncNodePurity'(decrease in node impurity), which is faster than '%IncMSE' (decrease in accuracy).

`trace` logical. To show the progress or not (default).

`minR` numeric. The minimum correlation coefficient of prediction is to control model accuracy. Default is 0.3.

`enrichTest` character, specifying the enrichment analysis method, which is either 'FET' (Fisher's exact test) or 'GSEA' (gene set enrichment analysis).

`namedScoresCutoffs`
numeric, the significance cutoff for the differential analysis p value. Default is 0.05.

`minSize` The minimum number (default 5) of target genes.

`maxSize` The maximum number (default 5000) of target genes.

`pvalueCutoff` numeric, the significance cutoff for adjusted enrichment p value. This is used for obtaining the 'topResult' slot in the final 'Enrich' object. Default is 0.05.

`qvalueCutoff` numeric, the significance cutoff of enrichment q-value. Default is 0.2.

`regAltName` alternative name for regulator. Default is NULL.

`universe` a vector of charactors. Background target genes.

`nperm` integer, number of permutations. The minimial possible nominal p-value is about 1/nperm. Default is 10000.

Value

an object of RegenrichSet class.

Examples

```
# library(RegEnrich)
data("Lyme_GSE63085")
data("TFs")

data = log2(Lyme_GSE63085$FPKM + 1)
colData = Lyme_GSE63085$sampleInfo
```

```
# Take first 2000 rows for example
data1 = data[seq(2000), ]

design = model.matrix(~0 + patientID + week, data = colData)

# Initializing a 'RegenrichSet' object
object = RegenrichSet(expr = data1,
                      colData = colData,
                      method = 'limma', minMeanExpr = 0,
                      design = design,
                      contrast = c(rep(0, ncol(design) - 1), 1),
                      networkConstruction = 'COEN',
                      enrichTest = 'FET')
object
```

RegenrichSet-class *RegenrichSet class*

Description

The RegenrichSet is the fundamental class that RegEnrich package is working with.

Slots

assayRaw matrix, the initial raw expression data.

colData DataFrame object, indicating sample information. Each row represent a sample and each column represent a feature of samples.

assays SimpleList object, containing the expression data after filtering (and after Variance Stabilizing Transformation, i.e. VST, if the differential analysis method is 'Wald_DESeq2' or 'LRT_DESeq2').

elementMetadata DataFrame object, a slot for saving results by differential expression analysis, containing at least three columns:'gene', 'p' and 'logFC'.

topNetwork TopNetwork object, a slot for saving top network edges. After regulator-target network inference, a TopNetwork-class object is assigned to this slot, containing only top ranked edges in the full network. Default is NULL.

resEnrich Enrich object, a slot for saving enrichment analysis either by Fisher's exact test (FET) or gene set enrichment analysis (GSEA).

resScore Score object, a slot for saving regulator ranking results. It contains five components, which are 'reg' (regulator), 'negLogPDEA' (-log10(p values of differential expression analysis)), 'negLogPEnrich' (-log10(p values of enrichment analysis)), 'logFC' (log2 fold changes), and 'score' (RegEnrich ranking score).

paramsIn list. The parameters used in the whole RegEnrich analysis. This slot can be updated by respecifying arguments in each step of RegEnrich analysis.

`paramsOut` a list of four elements: DeaMethod (differential expression method), networkType (regulator-target network construction method), percent (what percentage of edges from the full network is used), and enrichTest (enrichment method). By default, each element is NULL.

`network` `TopNetwork` object, a slot for saving a full network.

`regenrich_diffExpr` *Differential expression analysis step*

Description

This is the first step of RegEnrich analysis. differential expression analysis by this function needs to be performed on a 'RegenrichSet' object.

Usage

```
regenrich_diffExpr(object, ...)

## S4 method for signature 'RegenrichSet'
regenrich_diffExpr(object, ...)
```

Arguments

`object`	a 'RegenrichSet' object, which is initialized by `RegenrichSet` function.
`...`	arguments for differential analysis. After constructing a 'RegenrichSet' object, all arguments for RegEnrich analysis have been initialized and stored in 'paramsIn" slot. while the arguments for differential analysis can be re-specified here. These arguments include 'method', 'minMeanExpr', 'design', 'reduced', 'contrast', 'coef', 'name', 'fitType', 'sfType', 'betaPrior', 'minReplicatesForReplace', 'useT', 'minmu', 'parallel', 'BPPARAM', 'altHypothesis', 'listValues', 'cooksCutoff', 'independentFiltering', 'alpha', 'filter', 'theta', 'filterFun', 'addMLE', 'blind', 'ndups', 'spacing', 'block', 'correlation', 'weights', 'proportion', 'stdev.coef.lim', 'trend', 'robust', and 'winsor.tail.p'. See `RegenrichSet` function for more details about these arguments.

Value

This function returns a 'RegenrichSet' object with an updated 'resDEA' slot, which is a 'DeaSet' object, and an updated 'paramsIn' slot. See `newDeaSet` function for more details about 'DeaSet' class. If an argument not in the above list is specified in the regenrich_diffExpr function, a warning or error will be raised.

See Also

Initialization of a 'RegenrichSet' object `RegenrichSet`,and next step `regenrich_network`.

Examples

```
# library(RegEnrich)
data("Lyme_GSE63085")
data("TFs")

data = log2(Lyme_GSE63085$FPKM + 1)
colData = Lyme_GSE63085$sampleInfo

# Take first 2000 rows for example
data1 = data[seq(2000), ]

design = model.matrix(~0 + patientID + week, data = colData)

# Initializing a 'RegenrichSet' object
object = RegenrichSet(expr = data1,
                      colData = colData,
                      method = 'limma', minMeanExpr = 0,
                      design = design,
                      contrast = c(rep(0, ncol(design) - 1), 1),
                      networkConstruction = 'COEN',
                      enrichTest = 'FET')
# Using the predifined parameters in the previous step
(object = regenrich_diffExpr(object))

# re-specifying parameter 'minMeanExpr'
print(slot(object, 'paramsIn')$minMeanExpr)
(object = regenrich_diffExpr(object, minMeanExpr = 1))
print(slot(object, 'paramsIn')$minMeanExpr)

# Unrecognized argument 'unrecognizedArg' (Error)
# object = regenrich_diffExpr(object, minMeanExpr = 1,
#                             unrecognizedArg = 23)

# Argument not for differential expression analysis (Warning)
# print(slot(object, 'paramsIn')$networkConstruction)
# (object = regenrich_diffExpr(object, minMeanExpr = 1,
#                              networkConstruction = 'GRN'))
# print(slot(object, 'paramsIn')$networkConstruction) # not changed
```

---

regenrich_enrich *Enrichment analysis step*

Description

As the thrid step of RegEnrich analysis, enrichment analysis is followed by differential expression analysis (regenrich_diffExpr), and regulator-target network inference (regenrich_network).

Usage

```
regenrich_enrich(object, ...)

## S4 method for signature 'RegenrichSet'
regenrich_enrich(object, ...)
```

Arguments

object	a ‘RegenrichSet‘ object, to which regenrich_diffExpr, and regenrich_network, functions have been already applied.
...	arguments for enrichment analysis. After constructing a ‘RegenrichSet‘ object using RegenrichSet function, all arguments for RegEnrich analysis have been initialized and stored in ‘paramsIn“ slot. The arguments for enrichment analysis can be re-specified here. These arguments include ’enrichTest’, ’namedScoresCutoffs’, ’minSize’, ’maxSize’, ’pvalueCutoff’,’qvalueCutoff’, ’regAltName’, ’universe’, ’minSize’, ’maxSize’, ’pvalueCutoff’, and ’nperm’. See RegenrichSet function for more details about these arguments.

Value

This function returns a ‘RegenrichSet‘ object with an updated ‘resEnrich‘ slots, which is ‘Enrich‘ objects, and an updated ‘paramsIn‘ slot. See Enrich-class function for more details about ‘Enrich‘ class.

See Also

Previous step regenrich_network, and next step regenrich_rankScore.

Examples

```
# library(RegEnrich)
data("Lyme_GSE63085")
data("TFs")

data = log2(Lyme_GSE63085$FPKM + 1)
colData = Lyme_GSE63085$sampleInfo

# Take first 2000 rows for example
data1 = data[seq(2000), ]

design = model.matrix(~0 + patientID + week, data = colData)

# Initializing a 'RegenrichSet' object
object = RegenrichSet(expr = data1,
                      colData = colData,
                      method = 'limma', minMeanExpr = 0,
                      design = design,
```

```
                        contrast = c(rep(0, ncol(design) - 1), 1),
                        networkConstruction = 'COEN',
                        enrichTest = 'FET')


# Differential expression analysis
object = regenrich_diffExpr(object)

# Network inference using 'COEN' method
object = regenrich_network(object)

# Enrichment analysis by Fisher's exact test (FET)
(object = regenrich_enrich(object))

# Enrichment analysis by Fisher's exact test (GSEA)
(object = regenrich_enrich(object, enrichTest = "GSEA"))
```

---

regenrich_network *Regulator-target network inference step*

Description

As the second step of RegEnrich analysis, network inference is followed by differential expression analysis (regenrich_diffExpr).

Provide a network to ‘RegenrichSet‘ object.

Usage

```
regenrich_network(object, ...)

## S4 method for signature 'RegenrichSet'
regenrich_network(object, ...)

regenrich_network(object) <- value

## S4 replacement method for signature 'RegenrichSet,TopNetwork'
regenrich_network(object) <- value

## S4 replacement method for signature 'RegenrichSet,data.frame'
regenrich_network(object) <- value
```

Arguments

object	a ‘RegenrichSet‘ object, to which regenrich_diffExpr function has been already applied.

... arguments for network inference. After constructing a ‘RegenrichSet‘ object using RegenrichSet function, all arguments for RegEnrich analysis have been initialized and stored in ‘paramsIn“ slot. The arguments for network inference can be re-specified here.

These arguments include ’networkConstruction’, ’reg’, ’rowSample’, ’softPower’, ’networkType’, ’TOMDenom’, ’RsquaredCut’, ’edgeThreshold’, ’K’, ’nbTrees’, ’importanceMeasure’, ’trace’, ’BPPARAM’, ’minR’, ’topNetPercent’, and ’directed’.

See RegenrichSet function for more details about these arguments.

value either a ‘TopNetwork‘ object or ‘data.frame‘ object. If value is a ‘data.frame‘ object, then the number of columns of

Value

This function returns a ‘RegenrichSet‘ object with an updated ‘network‘ and ‘topNetP‘ slots, which are ‘TopNetwork‘ objects, and an updated ‘paramsIn‘ slot. See TopNetwork-class class for more details.

This function returns a ‘RegenrichSet‘ object with an updated ‘network‘ and ‘topNetP‘ slots, which are ‘TopNetwork‘ objects, and an updated ‘paramsIn‘ slot. See TopNetwork-class class for more details.

See Also

Previous step regenrich_diffExpr, and next step regenrich_enrich. User defined network regenrich_network<-

Examples

```
# library(RegEnrich)
data("Lyme_GSE63085")
data("TFs")

data = log2(Lyme_GSE63085$FPKM + 1)
colData = Lyme_GSE63085$sampleInfo

# Take first 2000 rows for example
data1 = data[seq(2000), ]

design = model.matrix(~0 + patientID + week, data = colData)

# Initializing a 'RegenrichSet' object
object = RegenrichSet(expr = data1,
                      colData = colData,
                      method = 'limma', minMeanExpr = 0,
                      design = design,
                      contrast = c(rep(0, ncol(design) - 1), 1),
                      networkConstruction = 'COEN',
                      enrichTest = 'FET')
```

```
# Differential expression analysis
(object = regenrich_diffExpr(object))

# Network inference using 'COEN' method
(object = regenrich_network(object))
```

---

regenrich_rankScore *Regulator scoring and ranking*

Description

As the fourth step of RegEnrich analysis, regulator ranking is followed by differential expression analysis (regenrich_diffExpr), regulator-target network inference (regenrich_network), and enrichment analysis (regenrich_enrich).

Usage

```
regenrich_rankScore(object)

## S4 method for signature 'RegenrichSet'
regenrich_rankScore(object)
```

Arguments

object	a ‘RegenrichSet‘ object, to which regenrich_diffExpr, regenrich_network, and regenrich_enrich functions all have been already applied.

Value

This function returns a ‘RegenrichSet‘ object with an updated ‘resScore‘ slots, which is a ‘regEnrichScore‘ (also ‘data.frame‘) object, and an updated ‘paramsIn‘ slot. In the ‘regEnrichScore‘ object there are five columns, which are ’reg’ (regulator), ’negLogPDEA’ (-log10(p values of differential expression analysis)), ’negLogPEnrich’ (-log10(p values of enrichment analysis), ’logFC’ (log2 fold changes), and ’score’ (RegEnrich ranking score).

See Also

Previous step regenrich_enrich.

Examples

```
# library(RegEnrich)
data("Lyme_GSE63085")
data("TFs")

data = log2(Lyme_GSE63085$FPKM + 1)
```

```
colData = Lyme_GSE63085$sampleInfo

# Take first 2000 rows for example
data1 = data[seq(2000), ]

design = model.matrix(~0 + patientID + week, data = colData)

# Initializing a 'RegenrichSet' object
object = RegenrichSet(expr = data1,
                      colData = colData,
                      method = 'limma', minMeanExpr = 0,
                      design = design,
                      contrast = c(rep(0, ncol(design) - 1), 1),
                      networkConstruction = 'COEN',
                      enrichTest = 'FET')


# Differential expression analysis
object = regenrich_diffExpr(object)

# Network inference using 'COEN' method
object = regenrich_network(object)

# Enrichment analysis by Fisher's exact test (FET)
object = regenrich_enrich(object)

# Regulators ranking
(object = regenrich_rankScore(object))
```

`results_expr` *Result accessor functions*

Description

- results_expr accesses raw expression data.
- results_DEA accesses results from differential expression analysis.
- results_topNet accesses results from network inference.
- retults_enrich accesses results from FET/GSEA enrichment analysis.
- results_score accesses results from regulator scoring and ranking.

Usage

```
results_expr(object)

results_DEA(object)

results_topNet(object)
```

```
results_enrich(object)

results_score(object)
```

Arguments

object	RegenrichSet object.

Value

results_expr retures an expression matrix.

results_DEA returns a list result of differentila analysis.

results_topNet returns a TopNetwork object.

results_enrich returns an Enrich object by either FET or GSEA method.

results_score returns an data frame of summarized ranking scores of regulators.

Examples

```
# library(RegEnrich)
data("Lyme_GSE63085")
data("TFs")

data = log2(Lyme_GSE63085$FPKM + 1)
colData = Lyme_GSE63085$sampleInfo

# Take first 2000 rows for example
data1 = data[seq(2000), ]

design = model.matrix(~0 + patientID + week, data = colData)

# Initializing a 'RegenrichSet' object
object = RegenrichSet(expr = data1,
                      colData = colData,
                      method = 'limma', minMeanExpr = 0,
                      design = design,
                      contrast = c(rep(0, ncol(design) - 1), 1),
                      networkConstruction = 'COEN',
                      enrichTest = 'FET')


# Differential expression analysis
object = regenrich_diffExpr(object)
results_expr(object)
results_DEA(object)

# Network inference using 'COEN' method
object = regenrich_network(object)
results_topNet(object)

# Enrichment analysis by Fisher's exact test (FET)
```

```
object = regenrich_enrich(object)
results_enrich(object)

# Regulators ranking
object = regenrich_rankScore(object)
results_score(object)
```

`Score-class` *Score class*

Description

‘Score‘ class inherits tibble ("tbl"). The objects of ‘Score‘ class are to store information of regulator ranking scores.

Usage

```
newScore(
  reg = character(),
  negLogPDEA = numeric(),
  negLogPEnrich = numeric(),
  logFC = numeric(),
  score = numeric()
)
```

Arguments

`reg`	character, regulator IDs.
`negLogPDEA`	numeric, -log(p_DEA).
`negLogPEnrich`	numeric, -log(p_Enrich).
`logFC`	numeric, log2 fold change.
`score`	numeric, RegEnrich ranking score.

Value

newScore function returns a `Score` object.

Slots

`names` character vector, containing "reg", "negLogPDEA", "negLogPEnrich", "logFC", and "score".

`.Data` a list of length 5, each elements corresponds to the `names` slots.

`row.names` character, regulators corresponding to `.Data` slot.

`.S3Class` character vector, containing "tbl_df", "tbl", "data.frame", indicating the classes that ‘Score‘ class inherits.

Examples

```
newScore()
newScore(letters[1:5], 1:5, 1:5, -2:2, seq(2, 1, len = 5))
```

---

show,DeaSet-method *methods of generic function "show"*

Description

methods of generic function "show"

Usage

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

## S4 method for signature 'TopNetwork'
show(object)

## S4 method for signature 'Enrich'
show(object)

## S4 method for signature 'Score'
show(object)

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

Arguments

object	one object of either DeaSet, TopNetwork, Enrich, Score, or RegenrichSet class.

Value

show returns an invisible original object.

Examples

```
x = newScore(letters[1:5], 1:5, 1:5, -2:2, seq(2, 1, len = 5))
show(x)
```

TFs *Human gene regulators*

Description

The transcrpiton factors and co-factors in humans are considered the regulators in RegEnrich. And these regulators are obtained from (Han et al. 2015; Marbach et al. 2016; and Liu et al. 2015).

Usage

```
data(TFs)
```

Format

An object of 2-column `data.frame`; The first column is ENSEMBL ID of gene regulators. The second column is gene name of gene regulators. The row name of this data frame is identical to the ENSEMBL ID column.

References

Han et al. (2015) Scientific Reports, 5:11432 (PubMed), Liu et al. (2015) Database, bav095 (PubMed), Marbach et al. (2016) Nature Methods, 13(4):366-70 (PubMed).

TopNetwork-class *TopNetwork class*

Description

The ‘TopNetwork‘ object is to store either a full network (the percentage of top edges is 100 between 0 to 10).

Slots

`element` tibble, the pool of targets in the network.

`set` tibble, the pool of valid regulators.

`elementset` tibble, regulator-target edges with edge weights. and the elements are regulators of the targets indicated by the element name.

`directed` logical, whether the network is directed.

`networkConstruction` character, by which method this network is constructed. Either ’COEN’ (coexpression network using WGCNA), or ’GRN’ (gene regulatory network using random forest), or ’new’ (a network provided by the user).

`percent` numeric, what percentage of the top edges are remained. The value must be between 0 (excluding) and 100 (including).

`active` character, which data table is activated, the default is "elementset".

Index

∗ **datasets**
Lyme_GSE63085, 7
TFs, 34
∗ **internal**
reexports, 15
%>% *(reexports)*, 15
%>%, *15*

adjacency, *12*, *21*

DeaSet-class, 2
DESeq, *19*
dim,TopNetwork-method, 3
duplicateCorrelation, *20*

eBayes, *21*
Enrich-class, 4

getResultsNames, 5
ggplot, *11*

head,Score-method, 6

importance, *22*

lmFit, *20, 21*
Lyme_GSE63085, 7

newDeaSet, 7, *24*
newScore *(Score-class)*, 32
newTopNetwork, 8

plot_Enrich, 13
plot_Enrich,RegenrichSet-method *(plot_Enrich)*, 13
plotOrders, 9
plotRegTarExpr, 10
plotSoftPower, 12, *21*
print.Score, 14

reexports, 15
regenrich_diffExpr, *5*, *10*, 24, *26–29*
regenrich_diffExpr,RegenrichSet-method *(regenrich_diffExpr)*, 24
regenrich_enrich, 25, *28, 29*
regenrich_enrich,RegenrichSet-method *(regenrich_enrich)*, 25
regenrich_network, *10*, *24*, *26*, 27, *29*
regenrich_network,RegenrichSet-method *(regenrich_network)*, 27
regenrich_network<- *(regenrich_network)*, 27
regenrich_network<-,RegenrichSet,data.frame-method *(regenrich_network)*, 27
regenrich_network<-,RegenrichSet,TopNetwork-method *(regenrich_network)*, 27
regenrich_rankScore, *26*, 29
regenrich_rankScore,RegenrichSet-method *(regenrich_rankScore)*, 29
RegenrichSet, 16, *24*, *26*, *28*
RegenrichSet-class, 23
results, *5*, *19, 20*
results_DEA *(results_expr)*, 30
results_enrich *(results_expr)*, 30
results_expr, 30
results_score *(results_expr)*, 30
results_topNet *(results_expr)*, 30
resultsNames, *5*

Score-class, 32
show,DeaSet-method, 33
show,Enrich-method *(show,DeaSet-method)*, 33
show,RegenrichSet-method *(show,DeaSet-method)*, 33
show,Score-method *(show,DeaSet-method)*, 33
show,TopNetwork-method *(show,DeaSet-method)*, 33

tail,Score-method *(head,Score-method)*, 6

TFs, 34
TopNetwork-class, 34

vst, *20*