Package ‘MIGSA’

April 26, 2017

Type Package
Title Massive and Integrative Gene Set Analysis
Version 1.0.0
Date 2016-03-02
Author Juan C. Rodriguez, Cristobal Fresno, Andrea S. Llera and Elmer A. Fernandez
Maintainer Juan C. Rodriguez <jcrodriguez@bdmg.com.ar>
Description Massive and Integrative Gene Set Analysis.
   The MIGSA package allows to perform a massive and integrative gene set
   analysis over several expression and gene sets simultaneously. It provides
   a common gene expression analytic framework that grants a comprehensive
   and coherent analysis. Only a minimal user parameter setting is required
   to perform both singular and gene set enrichment analyses in an
   integrative manner by means of the best available methods, i.e. dEnricher
   and mGSZrespectively.
   The greatest strengths of this big omics data tool are the availability of
   several functions to explore, analyze and visualize its results in order
   to facilitate the data mining task over huge information sources.
   MIGSA package also provides several functions that allow to easily load
   the most updated gene sets from several repositories.

URL http://www.bdmg.com.ar/
biocViews Software, GeneSetEnrichment, Visualization, GeneExpression,
   Microarray, RNASeq, KEGG
License GPL (>= 2)
Depends R (>= 3.4), methods, BiocGenerics
Suggests breastCancerMAINZ, breastCancerNKI, breastCancerTRANSBIG,
   breastCancerUNT, breastCancerUPP, breastCancerVDX, pbcmc,
   MIGSADATA
Imports AnnotationDbi, Biobase, BiocParallel, data.table, edgeR,
   futile.logger, ggplot2, GO.db, GOstats, gplots, graph,
   graphics, gridDevices, GSEABase, limma, matrixStats, mGSZ,
   org.Hs.eg.db, RBGL, Rgraphviz, RJJSONIO, stats, vegan
   ’SEAparams-class.R’ ‘SEAparams.R’ ’GSEAparams-class.R’
   ’GSEAparams.R’ ’FitOptions-class.R’ ’ExprData-class.R’
   ’IGSAinput-class.R’ ’IGSAinput.R’ ’FitOptions.R’ ’DEnricher.R’
as.Genesets

'Genesets-asGenesets.R' 'Genesets-enrichr.R'
'Genesets-geneSetsFromFile.R' 'Genesets-loadGo.R'
'GenesetsRes.R' 'GoAnalysis.R' 'MGSZ.R' 'IGSAs.R'
'IGSAInput-getterSetters.R' 'IGSA.R' 'IGSAInput-getDEGenes.R'
'MIGSAres-class.R' 'MIGSA.R' 'MIGSApackage.R' 'MIGSAmGSZ.R'
'MIGSAres.R' 'MIGSAres-GoAnalysis.R' 'MIGSAres-common.R'
'MIGSAres-setEnrCutoff.R' 'MIGSAres-genesManipulation.R'
'MIGSAres-getAdditionalInfo.R' 'MIGSAres-plotting.R'
'data-bcmigsaRes.R' 'data-migsaRes.R' 'summaryFunctions.R'

RoxygenNote 5.0.1.9000
NeedsCompilation no

R topics documented:

as.Genesets ................................. 2
bcMigsaRes .................................. 3
ExprData-class .............................. 4
FitOptions-class ............................ 5
Genesets-enrichr ........................... 6
geneSetsFromFile ......................... 7
getAdditionalInfo ......................... 8
getDEGenes ................................. 9
GSEAParamS-class ......................... 10
IGSAInput-class ......................... 11
IGSAInput-getterSetters ................ 12
loadGo ...................................... 14
MIGSA ...................................... 15
MIGSAmGSZ ................................ 16
MIGSApackage .............................. 18
migsaRes .................................. 19
MIGSAres-class ............................. 19
MIGSAres-common ......................... 20
MIGSAres-genes ......................... 22
MIGSAres-GOanalysis .................... 23
MIGSAres-plots ......................... 24
SEAParams-class ....................... 26
setEnrCutoff ......................... 27
summary .................................. 28

Index 30

as.Genesets

Create a GeneSetCollection from a list

Description

as.Genesets creates a GeneSetCollection object from the data present in a list. Each element will parse to a GeneSet. For each list element, its name will be the GeneSet setName, and the content are the genes.
Usage

as.Genesets(x, ...)  
## S4 method for signature 'list'
as.Genesets(x, is_GO = FALSE)

Arguments

x    list of character vectors which are the genes corresponding to each GeneSet. The list must have names (unique).
...
not in use.
is_GO logical indicating if this gene sets are from the Gene Ontology. If true, then each GeneSet setName must be a GO id.

Value

A GeneSetCollection object.

See Also

Genesets-enrichr
geneSetsFromFile
loadGo

Examples

## Lets create a list with three manually created gene sets and load it as a ## GeneSetCollection object.
myGs1 <- as.character(1:10);
myGs2 <- as.character(15:21);
myGs3 <- as.character(25:30);
myGssList <- list(myGs1, myGs2, myGs3);
names(myGssList) <- c("myGs1", "myGs2", "myGs3");
myGss <- as.Genesets(myGssList);

bcMigsaRes  A MIGSA results object obtained from breast cancer data

Description

A MIGSA results object (MIGSAres) obtained using the datasets:

- mainz (microarray).
- nki (microarray).
- tcga (microarray).
- tcga (RNAseq).
- transbig (microarray).
- unt (microarray).
• upp (microarray).
• vdx (microarray).

Each dataset subjects were classified using the PAM50 algorithm. For this analysis only Basal and Luminal A subjects were kept (this was the contrast used, i.e., Basal vs. LumA).

Format

A MIGSAres object.

Details

Datasets were tested for gene set enrichment over:

• the org.Hs.eg.db Gene Ontology gene sets, resulting in 14,291 as BP, 1,692 CC and 4,263 MF.
• KEGG gene sets downloaded from enrichr database resulting in 179 gene sets.

NOTE: For package space purposes, this bcMigsaRes object contains only the first 200 gene sets (subsetted from total result). For fully described information and a step-by-step guide in how this object was generated, please refer to MIGSA’s Vignette. Full bcMigsaRes object results can be downloaded (refer to Vignette).

ExprData-class

ExprData S4 class implementation in R

Description

This S4 class contains the expression data, it can be a MAList or DGEList. Important: Rownames are going to be used as gene identifiers, make sure that this IDs are the same used in your gene sets.

Examples

```r
## Lets create a ExprData object from counts data (DGEList).
rnaData <- matrix(rnbinom(10000, mu=5, size=2), ncol=4);
rownames(rnaData) <- 1:nrow(rnaData); # It must have rownames (gene names).

## Now we can use rnaExprData as an ExprData object.
## Not run:
rnaExprData <- DGEList(counts=rnaData);

## End(Not run)

## Lets create a ExprData object from micro array data (MAList).
maData <- matrix(rnorm(10000), ncol=4);
rownames(maData) <- 1:nrow(maData); # It must have rownames (gene names).

## Now we can use maExprData as an ExprData object.
maExprData <- new("MAList", list(M=maData));
```
FitOptions-class

Description

This S4 class contains the parameters to provide for model fitting. If the vector of samples is provided (must be two different, e.g. c("C1", "C1", "C2")) then it will contrast C1 vs. C2. If not, it should be provided with a data.frame x, the formula and the contrast, it will create the model matrix using x as data, and the formula.

Usage

FitOptions(x, ...)

## Default S3 method:
FitOptions(x, ...)

## S3 method for class 'data.frame'
FitOptions(x, formula, contrast, ...)

Arguments

x

There are two options for x:

• It can be a character vector containing the two conditions (length must be the same as the number of subjects to use).
• It can be a data.frame used as data by model.matrix.

...

not in use.

formula (only used if x is data.frame) used by model.matrix.

contrast (only used if x is data.frame) the contrast to test.

Value

FitOptions object.

Examples

## Suppose we have 15 subjects, the first 8 from Condition1 and the last 7
## from Condition2, lets create the corresponding FitOptions object to test
## Condition1 vs. Condition2.
l <- c(rep("Condition1", 8), rep("Condition2", 7));
fit_options <- FitOptions(l);

## Otherwise if we have the data and formula for model.matrix function and
## the desired contrast, we can create the FitOptions object as:
myData <- data.frame(cond=c(rep("Condition1", 8), rep("Condition2", 7)));
myFormula <- ~cond - 1;
myContrast <- c(-1, 1);
fit_options <- FitOptions(myData, myFormula, myContrast);
Genesets-enrichr

List and download gene sets from enrichr database

Description

enrichrGeneSets lists the database names present at enrichr. downloadEnrichrGeneSets creates a list of GeneSetCollection objects downloading the specified ones from enrichr website (http://amp.pharm.mssm.edu/Enrichr/).

Usage

"Genesets-enrichr"(pattern)

enrichrGeneSets(pattern = ".*")

## S4 method for signature 'ANY'

enrichrGeneSets(pattern = ".*")

downloadEnrichrGeneSets(geneSetNames, ...)

## S4 method for signature 'character'

downloadEnrichrGeneSets(geneSetNames, deleteMultipleEntrez = !FALSE)

Arguments

pattern character indicating a pattern to filter the database names.
geneSetNames list of characters with the names of the gene sets to download. Must be listed at enrichrGeneSets.
... not in use.
deleteMultipleEntrez logical indicating if multiple Entrez IDs should be deleted or repeated. Note: Enrichr uses Gene Symbol, if org.Hs.eg translates it into several Entrez IDs then if deleteMultipleEntrez == FALSE all Entrez are removed else all are included.

Value

enrichrGeneSets: character with present database names. downloadEnrichrGeneSets: list of GeneSetCollection objects (Genes are with their EntrezGene ID).

See Also

as.Genesets
geneSetsFromFile
loadGo
Examples

```r
## Lets list all the gene sets that can be downloaded from Enichr website.
enrichrGeneSets();

## Not run:
## Now lets list only the gene sets that have BioCarta in their names
## (different BioCarta versions).
enrichrGeneSets("BioCarta");

## End(Not run)

## And lets download the latest BioCarta gene sets from Enrichr.
## Make sure you use the same names as listed with enrichrGeneSets().
biocartaGSs <- downloadEnrichrGeneSets(c("BioCarta_2015"));
```

### geneSetsFromFile

**Description**

geneSetsFromFile creates a GeneSetCollection object from the data present in a file. The file must be a tab separated values file (tsv). Each line will parse to a GeneSet. First field will be the GeneSet setName, the second the setIdentifier and the remaining are the genes.

**Usage**

geneSetsFromFile(filePath, ...)

```r
## S4 method for signature 'character'
geneSetsFromFile(filePath, is_GO = FALSE)
```

**Arguments**

- **filePath** character with the path of the file to parse.
- **is_GO** logical indicating if this gene sets are from the Gene Ontology. If true, then each gene GeneSet setName must be a GO id.

**Value**

A GeneSetCollection object.

**See Also**

- `as.Genesets`
- `Genesets-enrichr`
- `loadGo`
Examples

```r
## Create some fake gene sets in a data.frame to save them in disk and then
## load them (10 gene sets with 20 genes each (it is not necesary that they
## have the same number of genes).
genets <- data.frame(
  IDs=paste("set", 1:10),
  Names=rep("", 10),
  matrix(paste("gene", 1:(10*20)), nrow=10));
## And save this file as a tab separated file.
genesFile <- paste(tempdir(), "/fakeGsets.tsv", sep="");
write.table(gsets, file=genesFile, sep="\t",
  col.names=FALSE, row.names=FALSE, quote=FALSE);
## Now lets load this tsv file as a GeneSetCollection object.
myGsets <- geneSetsFromFile(genesFile);
## And lets delete this tsv file (so we dont have garbage in our disk).
unlink(genesFile);
```

getAdditionalInfo  Gets additional information about enrichment results

Description

getAdditionalInfo gets additional enrichment information of the analysis done.

Usage

```r
getAdditionalInfo(migsaRes)
## S4 method for signature 'MIGSAres'
getAdditionalInfo(migsaRes)
```

Arguments

```r
migsaRes  MIGSAres object.
```

Value

data.frame with additional information for each analyzed gene set.

See Also

```r
setEnrCutoff
```
**getDEGenes**

*Calculate differentially expressed genes of an IGSAinput object*

**Description**

`getDEGenes` calculates the differentially expressed genes of an IGSAinput object using the expression matrix, the `FitOptions` and the `SEApars`...

**Usage**

```r
getDEGenes(igsaInput)
```

## S4 method for signature 'IGSAinput'

```r
gtDEGenes(igsaInput)
```

**Arguments**

- `igsaInput`: a valid IGSAinput object.

**Value**

A IGSAinput object with the updated slots and DE genes calculated.

**See Also**

- `IGSAinput-class`

**Examples**

```r
## Lets create a basic IGSAinput object.
## First create a expression matrix.
maData <- matrix(rnorm(10000), ncol=4);
rownames(maData) <- 1:nrow(maData); # It must have rownames (gene names).
maExprData <- new("MAList", list(M=maData));

## Now lets create the `FitOptions` object.
myFOpts <- FitOptions(c("Cond1", "Cond1", "Cond2", "Cond2"));
```
## Let's create our IGSAinput object.
myIgsaInput <- IGSAinput(name="myIgsaInput", expr_data=maExprData, fit_options=myFOpts);

## And check how many differentially expressed genes does it get with actual ## parameters.
aux <- getDEGenes(myIgsaInput);

---

**GSEAp-params-class**

*GSEAp-params S4 class implementation in R*

**Description**

This S4 class contains the parameters to provide for GSEA.

**Slots**

- `perm_number` number of permutations for p-value calculation (default: 200).
- `min_sz` minimum size of gene sets (number of genes in a gene set) to be included in the analysis (default: 5).
- `pv` estimate of the variance associated with each observation (default: 0).
- `w1` weight 1, parameter used to calculate the prior variance obtained with class size var.constant. This penalizes especially small classes and small subsets. Values around 0.1 - 0.5 are expected to be reasonable. (default: 0.2).
- `w2` weight 2, parameter used to calculate the prior variance obtained with the same class size as that of the analyzed class. This penalizes small subsets from the gene list. Values around 0.3 and 0.5 are expected to be reasonable (default: 0.5).
- `vc` size of the reference class used with wgt1. (default: 10).

**See Also**

- `SEAp-params-class`
- `summary`

**Examples**

## Let's create the default GSEAp-params object.
myGseaParams <- GSEAp-params();

## Let's create another GSEAp-params object with 500 permutations.
myGseaParams500Perms <- GSEAp-params(perm_number=500);
Description

This S4 class contains all the necessary inputs to execute a functional analysis (SEA and GSEA) on one experiment. Important: Make sure that gene IDs are concordant between the expression matrix and the provided gene sets.

Slots

name character indicating the name of this experiment.
expr_data ExprData object with the expression data (MicroArray or RNAseq). Note: expr_data can be a 0x0 matrix only if gsea_params=NULL and de_genes and br slots from sea_params are correctly set (vectors of gene names), in this case only SEA will be run.
fit_options FitOptions object with the parameters to be used when fitting the model.
gene_sets_list named list of GeneSetCollection objects to be tested for enrichment (names must be unique).
sea_params SEApars object with the parameters to be used by SEA, if NULL then SEA wont be run.
gsea_params GSEApars object with the parameters to be used by GSEA, if NULL then GSEA wont be run.

See Also

ExprData-class
SEApars-class
GSEApars-class
IGSAinput-getterSetters
getDEGenes
MIGSA
summary

Examples

## Lets create a basic IGSAinput object.
## First create a expression matrix.
maData <- matrix(rnorm(10000),ncol=4);
rownames(maData) <- 1:nrow(maData); # It must have rownames (gene names).
maExprData <- new("MAList",list(M=maData));

## Now lets create the FitOptions object.
myFOpts <- FitOptions(c("Cond1", "Cond1", "Cond2", "Cond2"));

## Finally lets create the Genesets to test for enrichment.
library(GSEABase);
myGs1 <- GeneSet(as.character(1:10), setIdentifier="fakeId1", setName="fakeName1");
myGs2 <- GeneSet(as.character(7:15), setIdentifier="fakeId2");
setName="fakeName2");
myGSs <- GeneSetCollection(list(myGs1, myGs2));

## And now we can create our IGSAinput ready for MIGSA.
igsaInput <- IGSAinput(name="igsaInput", expr_data=maExprData,
fit_options=myFOpts, gene_sets_list=list(myGSs=myGSs));

## Valid IGSAinput object with no expr_data (only run SEA).
igsaInput <- IGSAinput(name="igsaInput", gene_sets_list=list(myGSs=myGSs),
gsea_params=NULL,
sea_params=SEAprams(de_genes=rownames(maExprData)[1:10],
br=rownames(maExprData)));
validObject(igsaInput);

---

**IGSAinput-getterSetters**

*Accessors for IGSAinput class*

**Description**

Getters and setters functions to access IGSAinput object slots.

**Usage**

"IGSAinput-getterSetters"(object)

name(object)

## S4 method for signature 'IGSAinput'
name(object)

name(object) <- value

## S4 replacement method for signature 'IGSAinput'
name(object) <- value

fitOptions(object)

## S4 method for signature 'IGSAinput'
fitOptions(object)

fitOptions(object) <- value

## S4 replacement method for signature 'IGSAinput'
fitOptions(object) <- value

exprData(object)

## S4 method for signature 'IGSAinput'
exprData(object)
exprData(object) <- value

## S4 replacement method for signature 'IGSAinput'
exprData(object) <- value

geneSetsList(object)

## S4 method for signature 'IGSAinput'
geneSetsList(object)

geneSetsList(object) <- value

## S4 replacement method for signature 'IGSAinput'
geneSetsList(object) <- value

gseaParams(object)

## S4 method for signature 'IGSAinput'
gseaParams(object)
gseaParams(object) <- value

## S4 replacement method for signature 'IGSAinput'
gseaParams(object) <- value

seaParams(object)

## S4 method for signature 'IGSAinput'
seaParams(object)

seaParams(object) <- value

## S4 replacement method for signature 'IGSAinput'
seaParams(object) <- value

Arguments

object       IGSAinput object.
value     value to replace in the slot.

Value

Modified IGSAinput object or desired slot.

See Also

IGSAinput-class

Examples

## Lets create a basic IGSAinput object.  
## First create a expression matrix.
maData <- matrix(rnorm(10000),ncol=4);
rownames(maData) <- 1:nrow(maData); # It must have rownames (gene names).
maExprData <- new("MAList",list(M=maData));

## Now lets create the FitOptions object.
myFOpts <- FitOptions(c("Cond1", "Cond1", "Cond2", "Cond2"));

## And now we can create our IGSAinput ready for MIGSA.
igsaInput <- IGSAinput(name="igsaInput", expr_data=maExprData, 
fit_options=myFOpts);

## Lets get igsaInput values, and modify its name.
name(igsaInput);
name(igsaInput) <- "newName";
fitOptions(igsaInput);
exprData(igsaInput);
geneSetsList(igsaInput);
gseaParams(igsaInput);
seaParams(igsaInput);

### loadGo

Creates a GeneSetCollection object using the Gene Ontology data base

**Description**

loadGo creates a GeneSetCollection object from the data present at the Gene Ontology data base (org.Hs.eg.db R package).

**Usage**

loadGo(ontology)

## S4 method for signature 'character'
loadGo(ontology)

**Arguments**

- **ontology** character indicating which ontology must be loaded. Must be one of BP, MF or CC.

**Value**

A GeneSetCollection object(Genes are with their EntrezGene ID).

**See Also**

- as.Genesets
- Genesets-enrichr
- geneSetsFromFile
## Example

### Let's load the Cellular Components gene sets from the Gene Ontology.

```r
ccGSets <- loadGo("CC");
```

### MIGSA execution

#### Description

MIGSA runs a MIGSA execution. Functional analysis is done for each experiment by means of dEnricher and mGSZ.

#### Usage

```r
MIGSA(igsaInputs, ...)
```

#### Arguments

- `igsaInputs`: list of IGSAinput objects to execute.
- `...`: not in use.
- `geneSets`: (optional) named list of GeneSetCollection objects to be tested for enrichment (names must be unique). If provided then it will be tested in every IGSAinput, if not, each IGSAInput object must have its own list of GeneSetCollection.
- `bp_param`: (optional) BiocParallelParam to execute MIGSA in parallel.
- `saveResults`: logical indicating if back up of each individual experiment analysis should be saved. It will be saved in `getwd()"\migsaResults"/experimentName".RData". (default: FALSE).

#### Value

A MIGSAres object.

#### See Also

- `IGSAinput-class`
- `MIGSAres-class`

### Examples

#### Let's simulate two expression matrices of 1000 genes and 30 subjects.

```r
nGenes <- 1000; # 1000 genes
nSamples <- 30; # 30 subjects
geneNames <- paste("g", 1:nGenes, sep = ""); # with names g1 ... g1000
## Create random gene expression data matrix.
set.seed(8818);
exprData1 <- matrix(rnorm(nGenes*nSamples),ncol=nSamples);
```
rownames(exprData1) <- geneNames;
exprData2 <- matrix(rnorm(nGenes*nSamples),ncol=nSamples);
rownames(exprData2) <- geneNames;

## There will be 40 differentially expressed genes.
nDeGenes <- nGenes/25;
## Lets generate the offsets to sum to the differentially expressed genes.
deOffsets <- matrix(2*abs(rnorm(nDeGenes*nSamples/2)), ncol=nSamples/2);

## Randomly select which are the DE genes.
deIndexes1 <- sample(1:nGenes, nDeGenes, replace=FALSE);
exprData1[deIndexes1, 1:(nSamples/2)] <- exprData1[deIndexes1, 1:(nSamples/2)] + deOffsets;

deIndexes2 <- sample(1:nGenes, nDeGenes, replace=FALSE);
exprData2[deIndexes2, 1:(nSamples/2)] <- exprData2[deIndexes2, 1:(nSamples/2)] + deOffsets;

exprData1 <- new("MAList",list(M=exprData1));
exprData2 <- new("MAList",list(M=exprData2));

## 15 subjects with condition C1 and 15 with C2.
conditions <- rep(c("C1", "C2"),c(nSamples/2,nSamples/2));

nGSets <- 200; # 200 gene sets
## Lets create randomly 200 gene sets, of 10 genes each
gSets <- lapply(1:nGSets, function(i) sample(geneNames, size=10));
names(gSets) <- paste("set", as.character(1:nGSets), sep="";
myGSs <- as.Genesets(gSets);

fitOpts <- FitOptions(conditions);

igsaInput1 <- IGSAinput(name="igsaInput1", expr_data=exprData1, fit_options=fitOpts, gene_sets_list=list(myGSs=myGSs));
igsaInput2 <- IGSAinput(name="igsaInput2", expr_data=exprData2, fit_options=fitOpts, gene_sets_list=list(myGSs=myGSs));

experiments <- list(igsaInput1, igsaInput2);

## Finally run MIGSA!
## Not run:
migsaRes <- MIGSA(experiments);

## End(Not run)

---

**MIGSAmGSZ**

**Description**

MIGSAmGSZ is an optimized mGSZ version. It runs much faster than the original mGSZ version, moreover it can run in multicore technology. It allows to analyze RNAseq data by using **voom** function. mGSZ: Gene set analysis based on Gene Set Z scoring function and asymptotic p-value.
MIGSAmGSZ

Usage

MIGSAmGSZ(x, y, l, ...)  

## S4 method for signature 'matrix,list,vector'
MIGSAmGSZ(x, y, l, use.voom = FALSE,  
bp.param = bpparam(), rankFunction = NA, min.sz = 5, pv = 0,  
w1 = 0.2, w2 = 0.5, vc = 10, p = 200)

Arguments

x      gene expression data matrix (rows as genes and columns as samples).
y      gene set data (list).
l      vector of response values (example:c("Cond1","Cond1","Cond2", "Cond2","Cond2")).
...     not in use.
use.voom logical indicating wether use voom or not (if RNAseq data we recommend using use.voom=TRUE).
bp.param BiocParallel param to use as BPPARAM in bplapply.
rankFunction internal use.
min.sz    minimum size of gene sets (number of genes in a gene set) to be included in the analysis.
pv      estimate of the variance associated with each observation.
w1      weight 1, parameter used to calculate the prior variance obtained with class size var.constant. This penalizes especially small classes and small subsets. Values around 0.1 - 0.5 are expected to be reasonable.
w2      weight 2, parameter used to calculate the prior variance obtained with the same class size as that of the analyzed class. This penalizes small subsets from the gene list. Values around 0.3 and 0.5 are expected to be reasonable.
vc      size of the reference class used with wgt1.
p      number of permutations for p-value calculation.

Value

A data.frame with gene sets p-values and additional information.

Examples

nGenes <- 1000; # 1000 genes
nSamples <- 30; # 30 subjects
geneNames <- paste("g", 1:nGenes, sep=""); # with names g1 ... g1000

## Create random gene expression data matrix.
set.seed(8818);
exprData <- matrix(rnorm(nGenes*nSamples),ncol=nSamples);
rownames(exprData) <- geneNames;

## There will be 40 differentialy expressed genes.
nDeGenes <- nGenes/25;
## Lets generate the offsets to sum to the differentialy expressed genes.
deOffsets <- matrix(2*abs(rnorm(nDeGenes*nSamples/2)), ncol=nSamples/2);

## Randomly select which are the DE genes.
deIndexes <- sample(1:nGenes, nDeGenes, replace=FALSE);
exprData[deIndexes, 1:(nSamples/2)] <-
exprData[deIndexes, 1:(nSamples/2)] + deOffsets;

## 15 subjects with condition C1 and 15 with C2.
conditions <- rep(c("C1", "C2"),c(nSamples/2,nSamples/2));

nGSets <- 200; # 200 gene sets
## Lets create randomly 200 gene sets, of 10 genes each
gSets <- lapply(1:nGSets, function(i) sample(geneNames, size=10));
names(gSets) <- paste("set", as.character(1:nGSets), sep="");

## Not run:
mGSZres <- MIGSAmGSZ(exprData, gSets, conditions);
## End(Not run)

---

### Description

The MIGSA package allows to perform a massive and integrative gene set analysis over several expression and gene sets simultaneously. It provides a common gene expression analytic framework that grants a comprehensive and coherent analysis. Only a minimal user parameter setting is required to perform both singular and gene set enrichment analyses in an integrative manner by means of the best available methods, i.e. dEnricher and mGSZ respectively. The greatest strengths of this big omics data tool are the availability of several functions to explore, analyze and visualize its results in order to facilitate the data mining task over huge information sources. MIGSA package also provides several functions that allow to easily load the most updated gene sets from several repositories.

### Author(s)

Juan C. Rodriguez <jcrodriguez@bdmg.com.ar>, Cristobal Fresno <cfresno@bdmg.com.ar>, Andrea S. Llera <allera@leloir.org.ar> and Elmer A. Fernandez <efernandez@bdmg.com.ar>

### References

A MIGSA results object (MIGSAres) example

MIGSAres-class

MIGSAres S4 class implementation in R

Description

This S4 class represents the results of one MIGSA execution in R.

See Also

MIGSA
getAdditionalInfo
MIGSAres-common
MIGSAres-genes
MIGSAres-GOanalysis
MIGSAres-plots
setEnrCutoff
summary
MIGSAres-common

MIGSAres exploratory functions

Description
Several R base overwritten functions to manipulate a MIGSAres object as a data.frame way. NOTE: When subsetting a MIGSAres object, if it does not have the id, GS_Name and (at least) one experiment columns, then it wont be a MIGSAres object, i.e., migsaRes[,c("id","igsalInput1")]) is no longer a MIGSAres object.

Usage

## S4 method for signature 'MIGSAres'
dim(x)

## S4 method for signature 'MIGSAres'
x$name

## S4 method for signature 'MIGSAres'
colnames(x)

## S4 method for signature 'MIGSAres'
head(x, n = 6L)

## S4 method for signature 'MIGSAres'
tail(x, n = 6L)

## S4 method for signature 'MIGSAres,ANY,ANY,ANY'
x[i, j, drop = FALSE]

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

## S4 method for signature 'MIGSAres'
as.data.frame(x)

## S4 method for signature 'MIGSAres,MIGSAres'
merge(x, y)

Arguments

x            MIGSAres object.
name         as used in $.
n            as used in head and tail.
i            as used in [.
j            as used in [.
drop        as used in [ (default: FALSE).
object      MIGSAres object.
y            MIGSAres object.
Value

Desired object.

Examples

```r
data(migsaRes);
## As we ran MIGSA for two experiments and 200 gene sets, it must have 200
## rows, and five columns (id, Name, GS_Name, and the experiments names).
dim(migsaRes);

## migsaRes shown as data.frame has these column names: id, Name, GS_Name,
## and the experiments names. As we ran two experiments, names igsaInput1
## and igsaInput2, we can use $ in these ways:
head(migsaRes$id);
table(migsaRes$Name);
table(migsaRes$GS_Name);
head(migsaRes$igsaInput1);
head(migsaRes$igsaInput2);

colnames(migsaRes);
head(migsaRes);

## Or see the first 10
head(migsaRes, n=10);
tail(migsaRes);

## Or see the last 10
tail(migsaRes, n=10);

## migsaRes shown as data.frame has these column names: id, Name, GS_Name,
## and the experiments names. As we ran two experiments, names igsaInput1
## and igsaInput2, we can use [ in these ways:

## Lets get the first 5 rows and 4 columns (the result is a MIGSAres object).
migsaRes[1:5, 1:4];
class(migsaRes[1:5, 1:4]);

## Lets get the experiments results. Note that this is not any more a
## MIGSAres object.
migsaRes[, c("igsaInput1", "igsaInput2")];
class(migsaRes[, c("igsaInput1", "igsaInput2")]);
migsaRes;
migsaResDFrame <- as.data.frame(migsaRes);
head(migsaResDFrame);
migsaRes1 <- migsaRes[,1:4];
migsaRes2 <- migsaRes[,c(1:3,5)];
migsaResMerged <- merge(migsaRes1, migsaRes2);
```
genesInSets returns a data.frame with gene sets as rows, genes as columns, and as value the number of experiments in which each gene contributed to enrich each gene set. If it was enriched only by SEA then it returns the genes that contributed in SEA. If it was enriched only by GSEA then it returns the genes that contributed in GSEA. If it was enriched by both then it returns the genes that contributed in both. filterByGenes returns a MIGSAres object with only the gene sets which resulted enriched by at least one gene from a provided list.

Usage

"MIGSAres-genes"(migsaRes)
gegenesInSets(migsaRes)
## S4 method for signature 'MIGSAres'
gegenesInSets(migsaRes)
filterByGenes(migsaRes, genes)
## S4 method for signature 'MIGSAres,character'
filterByGenes(migsaRes, genes)

Arguments

migsaRes MIGSAres object.

genes character vector of the interest genes for MIGSAres filtering.

Value

If genesInSets: a data.frame with the number of experiments in which each gene contributed to enrich each gene set. If filterByGenes: A MIGSAres object containing only the gene sets in which provided genes contributed to enrichment.

See Also

setEnrCutoff

Examples

data(migsaRes);

####### genesInSets
## First lets set a cutoff of 0.1
migsaResWCoff <- setEnrCutoff(migsaRes, 0.1);
gInSets <- genesInSets(migsaResWCoff);
class(gInSets); # matrix

## Now we can do stuff as check which genes enriched a gene set in all (two)
## experiments.
gInSets[rowSums(gInSets==2) > 0, colSums(gInSets==2) > 0];

### filterByGenes

Suppose we are interested in studying these genes:
```r
intGenes <- c("g10", "g91", "g388", "g742", "g874");
migsaResIntGenes <- filterByGenes(migsaResWCoff, intGenes);
```

Now in migsaResIntGenes we have the MIGSA results of the gene sets in which at least one gene of our list contributed to enrich.

### Description

migsaGoTree plots the GO tree/s present in migsaRes. getHeights returns the heights of given a list of ids (GO IDs).

### Usage

```r
"MIGSAres-GOanalysis"(migsaRes)
migsaGoTree(migsaRes)
```

### Arguments

- `migsaRes` : MIGSAres object. It must contain at least one GO gene set.
- `ids` : character vector indicating the queried GO ids.
- `...` : not in use.
- `minHeight` : logical indicating if the minimum or maximum height must be calculated. If it is FALSE then the longest path to the root is calculated, otherwise, the shortest path.

### Value

If migsaGoTree: A list with the used data to plot. If getHeights: A list with each term height.
Examples

```r
## Lets load breast cancer results.
data(bcMigsaRes);

#### migsaGoTree
## Get the first 40 Gene Ontology gene sets results from CC.
goRes <- bcMigsaRes[bcMigsaRes$GS_Name == "CC",];
fst40goRes <- goRes[1:40,];

## And lets plot the results GO trees.
## Not run:
aux <- migsaGoTree(fst40goRes);

#### getHeights
## Get the first 40 Gene Ontology gene sets IDs.
goIds <- bcMigsaRes[bcMigsaRes$GS_Name %in% c("BP", "CC", "MF"), "id"];
fst40goIds <- goIds[1:40,];

## Not run:
## And lets get the heights in the GO tree structure.
getHeights(fst40goIds);

## End(Not run)
```

MIGSAres-plots

Description

genesHeatmap plots a heatmap with the number of experiments in which each gene contributed to enrich each gene set. genesBarplot generates a barplot of the number of gene sets in which each gene contributed to enrich. Each gene set counts 1 regardless if it was enriched in many experiments. x-axis each gene, y-axis number of gene sets in which it contributed to enrich. migsaHeatmap plots the enrichment heatmap of the MIGSAres object. geneSetBarplot generates a barplot of the number of experiments in which each gene set was enriched. x-axis each gene set, y-axis times it was enriched (0 to #experiments).

Usage

```
"MIGSAres-plots"(migsaRes)
genesHeatmap(migsaRes, ...)
## S4 method for signature 'MIGSAres'
genesHeatmap(migsaRes, enrFilter = 0, gsFilter = 0, ...)
genesBarplot(migsaRes, ...)
```
## S4 method for signature 'MIGSAres'
genesBarplot(migsaRes, enrFilter = 0, gsFilter = 0)

migsaHeatmap(migsaRes, ...)

## S4 method for signature 'MIGSAres'
migsaHeatmap(migsaRes, enrFilter = 0, expFilter = 0,
col.dist = "jaccard", row.dist = col.dist, ...)

geneSetBarplot(migsaRes, ...)

## S4 method for signature 'MIGSAres'
geneSetBarplot(migsaRes, enrFilter = 0)

### Arguments

- **migsaRes**: MIGSAres object.
- **...**: In heatmap functions, ... are additional parameters passed to heatmap.2 function. In other functions it is not in use.
- **enrFilter**: numeric. Keep gene sets enriched in at least enrFilter experiments.
- **gsFilter**: numeric. Keep genes enriched in at least gsFilter gene sets.
- **expFilter**: numeric. Keep experiments which enriched at least expFilter gene sets.
- **col.dist**: character. Distance algorithm to be used in columns, passed to vegdist function.
- **row.dist**: character. Distance algorithm to be used in rows, passed to vegdist function.

### Value

In heatmap functions: A list returned by heatmap.2 function (plotted data). In other functions: A ggplot object used as graphic.

### See Also

- `setEnrCutoff`

### Examples

data(migsaRes);
## For this example lets work with the first 50 results
migsaRes <- migsaRes[1:50,];

#### genesHeatmap
## First lets set a cutoff of 0.1
migsaResWCol <- setEnrCutoff(migsaRes, 0.1);

## Lets check what genes contributed to enrich the highest number of gene
## sets (in more than one gene set).
genesHeatmap(migsaResWCol, gsFilter=1);

## Moreover we can keep gene sets which where enriched in more than
## enrFilter experiments. To do this, we can use the enrFilter parameter.

#### genesBarplot
## Lets set a cutoff of 0.01
migsaResWCoff <- setEnrCutoff(migsaRes, 0.01);

## Lets check what genes contributed to enrich the highest number of gene
## sets (in more than one gene set).
genesBarplot(migsaResWCoff, gsFilter=1);

## Moreover we can keep gene sets which where enriched in more than
## enrFilter experiments. To do this, we can use the enrFilter parameter.

#### migsaHeatmap
## Lets set a cutoff of 0.1
migsaResWCoff <- setEnrCutoff(migsaRes, 0.1);

## Lets visually check enriched gene sets shared between experiments.
migsaHeatmap(migsaResWCoff);

#### geneSetBarplot
## Lets set a cutoff of 0.1
migsaResWCoff <- setEnrCutoff(migsaRes, 0.1);

## Lets check in how many experiments each gene set was enriched (in more
## than one experiment).
geneSetBarplot(migsaResWCoff, enrFilter=1);

SEApars-class

### Description

This S4 class contains the parameters to provide for SEA.

#### Slots

treat_lfc numeric, lfc parameter passed to `treat` function (default: 0).
dcutoff numeric, cutoff value to define a gene as differentially expressed (default: 0.01).
adjust_method character, method parameter passed to `p.adjust` function (default: "fdr").
de_genes (optional) character vector of differentially expressed genes, if not provided it will be
filled using the other parameters.
br background reference to use, there are two possible options for this slot (default: "briii"):  
  - character indicating to use "bri" or "briii".
  - character vector indicating the genes to use as background reference.
test character indicating which test to use, it must be one of "FisherTest", "HypergeoTest" or
  "BinomialTest" (default: "FisherTest").

### See Also

`GSEApars-class`

`summary`
Examples

## Lets create the default SEAprams object.
mySeaParams <- SEAprams();

## Lets create another SEAprams object with my default br genes. Make sure
## these genes are present in the analyzed expression matrix rownames.
myBr <- as.character(1:10);
mySeaParamsOwnBr <- SEAprams(br=myBr);

## Lets create another SEAprams object with my default differentially
## expressed genes. Make sure these genes are present in the analyzed
## expression matrix rownames.
myDEGenes <- as.character(3:15);
mySeaParamsOwnDEG <- SEAprams(de_genes=myDEGenes);

## Lets create another SEAprams object changing the differentially
## expressed genes parameters.
mySeaParamsOwnParams <- SEAprams(treat_lfc=0.25, de_cutoff=0.05,
adjust_method="none");


setEnrCutoff

Set enrichment cutoff for a MIGSAres object

Description

setEnrCutoff sets the enrichment cutoff value for a MIGSAres object in order to get detailed enrichment results.

Usage

setEnrCutoff(object, newEnrCutoff)

## S4 method for signature 'MIGSAres,numeric'
setEnrCutoff(object, newEnrCutoff)

Arguments

object a MIGSAres object.
newEnrCutoff numeric value in range [0,1] or NA to set the new enrichment cutoff. If NA then enrichment cutoff is unset for object.

Value

A MIGSAres object with the updated slots.

Examples

data(migsaRes);
## After executing MIGSA, the MIGSAres object does not have an enrichment
## cutoff set, so it will be shown as follows:
head(migsaRes);
## Now lets set an enrichment cutoff, and then show again the object.
migsaResWCoff <- setEnrCutoff(migsaRes, 0.01);
head(migsaResWCoff);

## So now we can do stuff like check how many gene sets are enriched in both
## experiments, in each one, etc. Moreover now we can do different MIGSAres
## plots available in MIGSA package.
table(migsaResWCoff[, c("igsaInput1", "igsaInput2")]);

## And with different cutoffs.
migsaResWCoff <- setEnrCutoff(migsaRes, 0.1);
table(migsaResWCoff[, c("igsaInput1", "igsaInput2")]);

---

### Summary functions for some MIGSA classes

**Description**

R base summary overwritten functions to manipulate MIGSA objects.

**Usage**

```r
## S3 method for class 'SEAparsms'
summary(object, ...)

## S3 method for class 'GSEAparsms'
summary(object, ...)

## S3 method for class 'IGSAinput'
summary(object, ...)

## S3 method for class 'MIGSAres'
summary(object, ...)
```

**Arguments**

- `object`  
  SEAparams, GSEAparsms, IGSAinput or MIGSAres object.
  
- `...`  
  not in use.

**Value**

A summary of the object.

**Examples**

```r
## Lets get the summary of the default SEAparsms object
seaParams <- SEAparsms();
summary(seaParams);

## Lets get the summary of the default GSEAparsms object
gseaParams <- GSEAparsms();
summary(gseaParams);
```
## Let's create a basic valid IGSAInput object to get its summary.

First create an expression matrix.
```r
maData <- matrix(rnorm(10000), ncol=4);
rownames(maData) <- 1:nrow(maData); # It must have rownames (gene names).
maExprData <- new("MAList", list(M=maData));
```

Now let's create the FitOptions object.
```r
myFOpts <- FitOptions(c("Cond1", "Cond1", "Cond2", "Cond2"));
```

And now we can create our IGSAInput ready for MIGSA.
```r
igsaInput <- IGSAInput(name="myIgsaInput", expr_data=maExprData, fit_options=myFOpts);
summary(igsaInput);
```

Now let's get the summary of out migsaRes data object.
```r
data(migsaRes);
```

### As enrichment cutoff is not set then we will get for each experiment the number of enriched gene sets at different cutoff values.
```r
summary(migsaRes);
```

### Let's set the enrichment cutoff at 0.01
```r
migsaResWCOff <- setEnrCutoff(migsaRes, 0.01);
```

### Now as summary we will get the number of enriched gene sets per experiment and their intersections.
```r
summary(migsaResWCOff);
```
Index

*Topic Analysis
  MIGSAPackage, 18

*Topic Functional
  MIGSAPackage, 18
[ , 20
[ , MIGSAres, ANY, ANY, ANY-method
  (MIGSAres-common), 20
$ , 20
$, MIGSAres-method (MIGSAres-common), 20
as.data.frame, MIGSAres
  (MIGSAres-common), 20
as.data.frame, MIGSAres-method
  (MIGSAres-common), 20
as.Genesets, 2, 6, 7, 14
as.Genesets, list-method (as.Genesets), 2
bcMigsAres, 3
colnames, MIGSAres
  (MIGSAres-common), 20
colnames, MIGSAres-method
  (MIGSAres-common), 20
dim, MIGSAres-method
  (MIGSAres-common), 20
downloadEnrichrGeneSets
  (Genesets-enrichr), 6
downloadEnrichrGeneSets, ANY-method
  (Genesets-enrichr), 6
downloadEnrichrGeneSets, character-method
  (Genesets-enrichr), 6
enrichrGeneSets, 6
enrichrGeneSets, Genesets-enrichr, 6
enrichrGeneSets, ANY-method
  (Genesets-enrichr), 6
enrichrGeneSets, character-method
  (Genesets-enrichr), 6
exprData, IGS Ainput-getterSetters, 12
exprData, IGS Ainput-method
  (IGSAinput-getterSetters), 12
ExprData-class, 4
exprData<-, IGS Ainput-getterSetters, 12
exprData<-, IGS Ainput-method
  (IGSAinput-getterSetters), 12
filterByGenes (MIGSAres-genes), 22
filterByGenes, MIGSAres, character-method
  (MIGSAres-genes), 22
FitOptions (FitOptions-class), 5
fitOptions (IGSAinput-getterSetters), 12
fitOptions, IGS Ainput-method
  (IGSAinput-getterSetters), 12
FitOptions-class, 5
FitOptions.data.frame
  (FitOptions-class), 5
FitOptions.default (FitOptions-class), 5
fitOptions<-, IGS Ainput-method
  (IGSAinput-getterSetters), 12
genesBarplot (MIGSAres-plots), 24
genesBarplot, MIGSAres-method
  (MIGSAres-plots), 24
geneSetBarplot (MIGSAres-plots), 24
geneSetBarplot, MIGSAres-method
  (MIGSAres-plots), 24
Genesets-enrichr, 6
geneSetsFromFile, 3, 6, 7, 14
geneSetsFromFile, character-method
  (geneSetsFromFile), 7
geneSetsList (IGSAinput-getterSetters), 12
geneSetsList, IGS Ainput-method
  (IGSAinput-getterSetters), 12
geneSetsList<-, (IGSAinput-getterSetters), 12
genexHeatmap (MIGSAres-plots), 24
genexHeatmap, MIGSAres-method
  (MIGSAres-plots), 24
genexInSets (MIGSAres-genes), 22
genexInSets, MIGSAres-method
  (MIGSAres-genes), 22
getAdditionalInfo, 8, 19
getAdditionalInfo, MIGSAres
  (getAdditionalInfo), 8