topGO

April 19, 2009

`GOFisherTest` *Fischer's exact test for gene set*

Description

~~ A concise (1-5 lines) description of what the function does. ~~

Usage

```
GOFisherTest(object)
```

Arguments

`object`	~~Describe `object` here~~

Details

fisher test based on contingency table

Value

p-value ...

Author(s)

Adrian Alexa

See Also

`GOKSTest`, `groupStats-class`, `getSigGroups-methods`

`GOKSTest` *Kolmogorov-Smirnov test for gene sets*

Description

~~ A concise (1-5 lines) description of what the function does. ~~

Usage

```
GOKSTest(object)
GOtTest(object)
```

Arguments

`object`	~~Describe `object` here~~

Details

~~ If necessary, more details than the description above ~~

Value

returns the p-value of the Running-Sum-Statistic

Author(s)

Adrian Alexa

See Also

`GOKSTest`, `groupStats-class`, `getSigGroups-methods`

`GOdata` *Example of a topGO data object*

Description

This data set contains a list of example `graphNEL` objects, which can then be used for plotting.

Usage

```
data(GOdata)
```

Source

Generated using the `ALL` gene expression data. See `topGOdata-class` for code examples on how-to generate this object.

Examples

```
data(GOdata)

## print the object
GOdata
```

`GOglobalTest` *Globaltest for gene sets*

Description

Warping finction for globaltest.

Usage

```
GOglobalTest(object)
```

Arguments

`object`	~~Describe `object` here~~

Details

~~ If necessary, more details than the description above ~~

Value

p-value of the ....

Author(s)

Adrian Alexa

See Also

`GOKSTest`, `groupStats-class`, `getSigGroups-methods`

`annFUN` *functions to map gene IDs to GO terms*

Description

These functions are used to compile a list of GO terms and their mappings to gene identifiers.

Usage

```
annFUN.db(whichOnto, feasibleGenes = NULL, affyLib)
annFUN(whichOnto, feasibleGenes = NULL, affyLib)
annFUN.gene2GO(whichOnto, feasibleGenes = NULL, gene2GO)
annFUN.GO2genes(whichOnto, feasibleGenes = NULL, GO2genes)
```

Arguments

`whichOnto`	character string specifying one of the three GO ontologies: `"BP"`, `"MF"`, `"CC"`
`feasibleGenes`	character vector containing a subset of gene identifiers. Only these genes will be used to annotate GO terms. Default value is `NULL` which means all gene identifiers will be used.
`affyLib`	character string containing the name of the Affymetrix chip.
`gene2GO`	named list of character vectors. The list names are genes identifiers. For each gene the character vector contains the GO terms IDs it maps to. Only the most specific annotations are required.
`GO2genes`	named list of character vectors. The list names are GO terms IDs. For each GO the character vector contains the genes identifiers which are mapped to it. Only the most specific annotations are required.

Details

The function `annFUN.db` uses the mappings provided in the Bioconductor annotation data packages. For example, if the Affymetrix hgu133a chip it is used, then the user should set `affyLib = "hgu133a.db"`.

The functions `annFUN.gene2GO` and `annFUN.GO2genes` are used when the user provide his own annotations.

All these function restrict the GO terms to the ones belonging to the specified ontology.

Value

A named(GO terms IDs) list of character vectors.

Author(s)

Adrian Alexa

See Also

`topGOdata-class`

Examples

```
library(hgu133a.db)
set.seed(111)

## generate a gene list and the GO annotations
numGenes <- 50
selGenes <- sample(ls(hgu133aGO), numGenes)
gene2GO <- lapply(mget(selGenes, envir = hgu133aGO), names)
gene2GO[sapply(gene2GO, is.null)] <- NA

## the annotation for the first three genes
gene2GO[1:3]

## inverting the annotations
go2genes <- annFUN.gene2GO(whichOnto = "CC", gene2GO = gene2GO)
```

```
## generate a GO list with the genes annotations
numGO <- 30
selGO <- sample(ls(hgu133aGO2PROBE), numGO)
GO2gene <- lapply(mget(selGO, envir = hgu133aGO2PROBE), as.character)

GO2gene[1:3]

## select only the GO terms for a specific ontology
go2gene <- annFUN.GO2genes(whichOnto = "CC", GO2gene = GO2gene)
```

`buildGOgraph.topology` *buils GO graph starting from the most specific GO terms*

Description

This function is building the GO graph starting from the most specific terms. The structure of the GO graph is build recursively.

Usage

```
buildGOgraph.topology(knownNodes, whichOnto = "BP")
```

Arguments

`knownNodes`	character vector of GO terms
`whichOnto`	character string specifying one of the three GO ontologies: `"BP"`, `"MF"`, `"CC"`

Value

An object of class `graphNEL-class` is returned. The graph is directed (the edges are from leaves to the root) and it contains all GO terms specific to `whichOnto` ontology.

Author(s)

Adrian Alexa

See Also

`topGOdata-class`, `buildLevels`, `mapGenes2GOgraph`, `annFUN`

`Determines the levels of a Directed Acyclic Graph (DAG)` *Determines the levels of a Directed Acyclic Graph (DAG)*

Description

TODO: This function take the a directed graph and constructs a named vector which contain the level on which a node is. The root has level 1.

TODO: Find the root(roots) of the DAG

Usage

```
buildLevels(dag, root = NULL, leafs2root = TRUE)
getNoOfLevels(graphLevels)
getGraphRoot(dag, leafs2root = TRUE)
```

Arguments

`dag`	~~Describe `dag` here~~
`root`	~~Describe `root` here~~
`leafs2root`	The leafs2root parameter tell if the graph has edges directed from the leaves to the root, or vice-versa
`graphLevels`	~~Describe `graphLevels` here~~

Details

.....

Value

it returns a list containing:

`level2nodes`	Environment where the key is the level number with the value being the nodes on that level.
`nodes2level`	Environment where the key is the node label (the GO ID) and the value is the level on which that node lies.
`noOfLevels`	The number of levels
`noOfNodes`	The number of nodes

Author(s)

Adrian Alexa

See Also

`topGOdata-class`, `reverseArch`, `inducedGraph`

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--     or do  help(data=index)  for the standard data sets.
```

`classicCount-class` *Class "classicCount"*

Description

This class that extends the virtual class "groupStats" by adding a slot representing the significant members. ....

Details

TODO: Some datails here.....

Objects from the Class

Objects can be created by calls of the form `new("classicCount", testStatistic = "function", name = "character", allMembers = "character", groupMembers = "character", sigMembers = "character")`.

Slots

`significant`: Object of class `"integer"` ~~

`name`: Object of class `"character"` ~~

`allMembers`: Object of class `"character"` ~~

`members`: Object of class `"character"` ~~

`testStatistic`: Object of class `"function"` ~~

Extends

Class `"groupStats"`, directly.

Methods

contTable `signature(object = "classicCount")`: ...

initialize `signature(.Object = "classicCount")`: ...

numSigAll `signature(object = "classicCount")`: ...

numSigMembers `signature(object = "classicCount")`: ...

sigAllMembers `signature(object = "classicCount")`: ...

sigMembers<- `signature(object = "classicCount")`: ...

sigMembers `signature(object = "classicCount")`: ...

Author(s)

Adrian Alexa

See Also

`classicScore-class`, `groupStats-class`, `getSigGroups-methods`

Examples

```
##---- Should be DIRECTLY executable !! ----
```

`classicExpr-class` *Class "classicExpr" ~~~*

Description

~~ A concise (1-5 lines) description of what the class is. ~~

Objects from the Class

Objects can be created by calls of the form `new("classicExpr", testStatistic, name, groupMembers, exprDat, pType, ...)`. ~~ describe objects here ~~

Slots

`eData`: Object of class `"environment"` ~~

`pType`: Object of class `"factor"` ~~

`name`: Object of class `"character"` ~~

`allMembers`: Object of class `"character"` ~~

`members`: Object of class `"character"` ~~

`testStatistic`: Object of class `"function"` ~~

`testStatPar`: Object of class `"list"` ~~

Extends

Class `"groupStats"`, directly.

Methods

allMembers<- `signature(object = "classicExpr")`: ...

emptyExpr `signature(object = "classicExpr")`: ...

getSigGroups `signature(object = "topGOdata", test.stat = "classicExpr")`: ...

GOglobalTest `signature(object = "classicExpr")`: ...

initialize `signature(.Object = "classicExpr")`: ...

membersExpr `signature(object = "classicExpr")`: ...

pType<- `signature(object = "classicExpr")`: ...

pType `signature(object = "classicExpr")`: ...

Author(s)

Adrian Alexa

See Also

`classicScore-class`, `groupStats-class`, `getSigGroups-methods`

Examples

```
showClass("classicExpr")
```

`classicScore-class` *Class "classicScore"*

Description

TODO: A class that extends the virtual class groupStats by adding a slot representing the score of each gene. (used for KS test)

Objects from the Class

Objects can be created by calls of the form `new("classicScore", testStatistic, name, allMembers, groupMembers, score, decreasing)`. ~~ describe objects here ~~

Slots

`score`: Object of class `"numeric"` ~~

`name`: Object of class `"character"` ~~

`allMembers`: Object of class `"character"` ~~

`members`: Object of class `"character"` ~~

`testStatistic`: Object of class `"function"` ~~

Extends

Class `"groupStats"`, directly.

Methods

allScore Method to obtain the score of all members.

scoreOrder Returns TRUE if the score should be ordered increasing, FALSE otherwise.

membersScore `signature(object = "classicScore")`: ...

rankMembers `signature(object = "classicScore")`: ...

score<- `signature(object = "classicScore")`: ...

Author(s)

Adrian Alexa

See Also

`classicCount-class`, `groupStats-class`, `getSigGroups-methods`

Examples

```
## define the type of test you want to use
test.stat <- new("classicScore", testStatistic = GOKSTest, name = "KS tests")
```

---

elimCount-class *Classes "elimCount" and "removeCount"*

Description

~~ A concise (1-5 lines) description of what the class is. ~~

Details

TODO: Some datails here.....

Objects from the Class

Objects can be created by calls of the form `new("elimCount", testStatistic, name, allMembers, groupMembers, sigMembers, elim, cutOff, ...)`. ~~ describe objects here ~~

Slots

`elim`: Object of class `"integer"` ~~

`cutOff`: Object of class `"numeric"` ~~

`significant`: Object of class `"integer"` ~~

`name`: Object of class `"character"` ~~

`allMembers`: Object of class `"character"` ~~

`members`: Object of class `"character"` ~~

`testStatistic`: Object of class `"function"` ~~

`testStatPar`: Object of class `"list"` ~~

Extends

Class `"classicCount"`, directly. Class `"groupStats"`, by class "classicCount", distance 2.

Methods

No methods defined with class "elimCount" in the signature.

Author(s)

Adrian Alexa

See Also

`classicScore-class`, `groupStats-class`, `getSigGroups-methods`

Examples

```
##---- Should be DIRECTLY executable !! ----
```

`elimExpr-class` *Class "elimExpr" ~~~*

Description

~~ A concise (1-5 lines) description of what the class is. ~~

Details

TODO: Some datails here.....

Objects from the Class

Objects can be created by calls of the form `new("elimExpr", testStatistic, name, groupMembers, exprDat, pType, elim, cutOff, ...)`. ~~ describe objects here ~~

Slots

`cutOff`: Object of class `"numeric"` ~~

`elim`: Object of class `"integer"` ~~

`eData`: Object of class `"environment"` ~~

`pType`: Object of class `"factor"` ~~

`name`: Object of class `"character"` ~~

`allMembers`: Object of class `"character"` ~~

`members`: Object of class `"character"` ~~

`testStatistic`: Object of class `"function"` ~~

`testStatPar`: Object of class `"list"` ~~

Extends

Class `"removeExpr"`, directly. Class `"classicExpr"`, by class "removeExpr", distance 2. Class `"groupStats"`, by class "removeExpr", distance 3.

Methods

cutOff<- `signature(object = "elimExpr")`: ...

cutOff `signature(object = "elimExpr")`: ...

getSigGroups `signature(object = "topGOdata", test.stat = "elimExpr")`: ...

initialize `signature(.Object = "elimExpr")`: ...

Author(s)

Adrian Alexa

See Also

`classicScore-class`, `groupStats-class`, `getSigGroups-methods`

Examples

```
showClass("elimExpr")
```

`elimScore-class` *Classes "elimScore" and "removeScore"*

Description

~~ A concise (1-5 lines) description of what the class is. ~~

Details

TODO:

Objects from the Class

Objects can be created by calls of the form `new("elimScore", testStatistic, name, allMembers, groupMembers, score, alternative, elim, cutOff, ...)`. ~~ describe objects here ~~

Slots

`elim`: Object of class `"integer"` ~~

`cutOff`: Object of class `"numeric"` ~~

`score`: Object of class `"numeric"` ~~

`.alternative`: Object of class `"logical"` ~~

`name`: Object of class `"character"` ~~

`allMembers`: Object of class `"character"` ~~

`members`: Object of class `"character"` ~~

`testStatistic`: Object of class `"function"` ~~

`testStatPar`: Object of class `"list"` ~~

Extends

Class `"classicScore"`, directly. Class `"groupStats"`, by class "classicScore", distance 2.

Methods

No methods defined with class "elimScore" in the signature.

Author(s)

Adrian Alexa

See Also

`classicScore-class`, `groupStats-class`, `getSigGroups-methods`

Examples

```
##---- Should be DIRECTLY executable !! ----
```

getPvalues *Function to compute p-values of a t-test for a gene expression matrix.*

Description

Warping function for computing the p-vales for a gene expression matrix.

Usage

```
getPvalues(edata, classlabel, test = "t", alternative = c("greater", "two.sid
genesID = NULL, correction = c("none", "Bonferroni", "Holm", "Hochberg", "Sid
"BH", "BY")[8])
```

Arguments

edata	Gene expression matrix.
classlabel	The phenotype of the data
test	Which test statistic to use
alternative	The alternative of the test statistic
genesID	if a subset of genes is provided
correction	Multiple testing correction procedure

Details

~~ If necessary, more details than the description above ~~

Value

An named vector of p-values is returned.

Author(s)

Adrian Alexa

See Also

GOKSTest, groupStats-class, getSigGroups-methods

Examples

```
library(ALL)
data(ALL)

## discriminate B-cell from T-cell
classLabel <- as.integer(sapply(ALL$BT, function(x) return(substr(x, 1, 1) == 'T')))

## Differentially expressed genes
geneList <- getPvalues(exprs(ALL), classlabel = classLabel,
                       alternative = "greater", correction = "BY")

hist(geneList, 50)
```

`getSigGroups` *Algorithms for scoring GO terms*

Description

TODO: This function is use for dispatching each algorithm

Usage

```
getSigGroups(object, test.stat, ...)
```

Arguments

`object`	~~Describe `object` here~~
`test.stat`	~~Describe `test.stat` here~~
`...`	~~Describe `...` here~~

Details

~~ If necessary, more details than the description above ~~

Value

~Describe the value returned If it is a LIST, use

`comp1`	Description of 'comp1'
`comp2`	Description of 'comp2'

...

Author(s)

Adrian Alexa

See Also

`topGOdata-class`, `classicCount-class`, `classicScore-class`

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--	or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function(object, test.stat, ...) standardGeneric("getSigGroups")
```

`groupGOTerms` *~~function to do ... ~~*

Description

TODO: Function that split GOTERM in different ontologies. Every new environment contain only the terms from one of the ontologies 'BP', 'CC', 'MF'

Usage

```
groupGOTerms(where)
```

Arguments

`where`	The the environment where you wantto bind the results

Details

~~ If necessary, more details than the description above ~~

Value

~Describe the value returned If it is a LIST, use

`comp1`	Description of 'comp1'
`comp2`	Description of 'comp2'

...

Author(s)

Adrian Alexa

See Also

`topGOdata-class`, `GOTerm`

Examples

```
groupGOTerms()
```

`groupStats-class` *Class "groupStats"*

Description

A virtual class containing basic group (GO term) data: gene names, genes scores, etc...

Objects from the Class

A virtual Class: No objects may be created from it.

Slots

`name`: Object of class `"character"` ~~

`allMembers`: Object of class `"character"` ~~

`members`: Object of class `"character"` ~~

`testStatistic`: Object of class `"function"` ~~

Methods

allMembers<- `signature(object = "groupStats")`: ...

allMembers `signature(object = "groupStats")`: ...

initialize `signature(.Object = "groupStats")`: ...

members<- `signature(object = "groupStats")`: ...

members `signature(object = "groupStats")`: ...

Name<- `signature(object = "groupStats")`: ...

Name `signature(object = "groupStats")`: ...

numAllMembers `signature(object = "groupStats")`: ...

numMembers `signature(object = "groupStats")`: ...

runTest `signature(object = "groupStats")`: ...

testStatistic `signature(object = "groupStats")`: ...

Author(s)

Adrian Alexa

See Also

`classicCount-class`, `getSigGroups-methods`

Examples

```
##---- Should be DIRECTLY executable !! ----
```

`inducedGraph` *~~function to do ... ~~*

Description

TODO: Given a GO term (or a list of GO terms) this function is returning the subgraph induced by node.

Usage

```
inducedGraph(dag, startNodes)
```

Arguments

`dag`	~~Describe `dag` here~~
`startNodes`	~~Describe `startNodes` here~~

Details

~~ If necessary, more details than the description above ~~

Value

An object of class `graphNEL-class` is returned.

Author(s)

Adrian Alexa

See Also

`topGOdata-class`, `reverseArch`,

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--    or do  help(data=index)  for the standard data sets.
```

`mapGenes2GOgraph` *~~function to do ... ~~*

Description

TODO: This function builds for each node a vector containing all the genes/probes that can be annotated to that node. It starts with the nodes on the lowest level, and then pushes their genes to the parents/ancestors

Usage

```
mapGenes2GOgraph(dag, mostSpecificGOs, nodeLevel = buildLevels(dag, leafs2root =
```

Arguments

`dag`	~~Describe `dag` here~~
`mostSpecificGOs`	~~Describe `mostSpecificGOs` here~~
`nodeLevel`	~~Describe `nodeLevel` here~~

Details

~~ If necessary, more details than the description above ~~

Value

An object of class `graphNEL-class` is returned. The attribute of each node in the graph contains a mapping of the genes/probes.

Author(s)

Adrian Alexa

See Also

`topGOdata-class`, `buildLevels`, `buildGOgraph.topology`, `annFUN`

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--     or do  help(data=index)  for the standard data sets.
```

`parentChild-class` *Classes "parentChild" and "pC"*

Description

~~ A concise (1-5 lines) description of what the class is. ~~

Objects from the Class

Objects can be created by calls of the form `new("parentChild", testStatistic, name, groupMembers, parents, sigMembers, joinFun, ...)`. ~~ describe objects here ~~

Slots

`splitIndex`: Object of class `"integer"` ~~

`joinFun`: Object of class `"character"` ~~

`significant`: Object of class `"integer"` ~~

`name`: Object of class `"character"` ~~

`allMembers`: Object of class `"character"` ~~

`members`: Object of class `"character"` ~~

`testStatistic`: Object of class `"function"` ~~

`testStatPar`: Object of class `"list"` ~~

Extends

Class `"classicCount"`, directly. Class `"groupStats"`, by class "classicCount", distance 2.

Methods

allMembers<- `signature(object = "parentChild")`: ...

allMembers `signature(object = "parentChild")`: ...

allParents `signature(object = "parentChild")`: ...

getSigGroups `signature(object = "topGOdata", test.stat = "parentChild")`: ...

initialize `signature(.Object = "parentChild")`: ...

joinFun `signature(object = "parentChild")`: ...

numAllMembers `signature(object = "parentChild")`: ...

numSigAll `signature(object = "parentChild")`: ...

sigAllMembers `signature(object = "parentChild")`: ...

sigMembers<- `signature(object = "parentChild")`: ...

updateGroup `signature(object = "parentChild", name = "missing", members = "character")`: ...

Author(s)

Adrian Alexa

See Also

`classicCount-class`, `groupStats-class`, `getSigGroups-methods`

Examples

```
showClass("parentChild")
showClass("pC")
```

`printGenes-methods` *Summary for genes annotated to a GO term*

Description

Function to print summary for the top genes annotated to the specified GO term.

Methods

~~describe this method here

Author(s)

Adrian Alexa

See Also

`classicScore-class`, `groupStats-class`, `getSigGroups-methods`

`printGraph-methods` *~~ Methods for Function printGraph in Package 'topGO' ~~*

Description

~~ Methods for function `printGraph` in Package 'topGO' ~~

Methods

~~describe this method here

Author(s)

Adrian Alexa

See Also

`classicScore-class`, `groupStats-class`, `getSigGroups-methods`

Examples

```
##---- Should be DIRECTLY executable !! ----
```

`reverseArch` *~~function to do ... ~~*

Description

TODO: Simple function to invert the direction of edges in an directed graph. The returned graph is of class graphNEL. It can use either simple matrices or sparse matrices (SparseM library)

Usage

```
reverseArch(dirGraph, useAlgo = "sparse", useWeights = TRUE)
```

Arguments

`dirGraph`	The graph to be transformed
`useAlgo`	"sparse" or "normal"
`useWeights`	If weights should be used (if useAlgo = 'normal' that the weigths are used anyway)

Details

~~ If necessary, more details than the description above ~~

Value

An object of class `graphNEL-class` is returned.

Author(s)

Adrian Alexa

See Also

`buildLevels`, `mapGenes2GOgraph`, `inducedGraph`

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--    or do  help(data=index)  for the standard data sets.
```

`topGO-package` *Enrichment analysis for Gene Ontology*

Description

topGO package provides tools for testing GO terms while accounting for the topology of the GO graph. Different test statistics and different methods for eliminating local similarities and dependencies between GO terms can be implemented and applied.

Details

Package:	topGO
Type:	Package
Version:	1.0
Date:	2006-10-02
License:	What license is it under?

TODO: An overview of how to use the package, including the most important functions

Author(s)

Adrian Alexa, Jörg Rahnenführer

Maintainer: Adrian Alexa <alexa@mpi-inf.mpg.de>

References

Alexa A., Rahnenführer J., Lengauer T., Improved scoring of functional groups from gene expression data by decorrelating GO graph structure, Bioinformatics 22(13): 1600-1607, 2006

See Also

`topGOdata-class`, `groupStats-class`, `getSigGroups-methods`

`topGOdata-class` *Class "topGOdata"*

Description

TODO: The node attributes are environments containing the genes/probes annotated to the respective node

If genes is a numeric vector than this should represent the gene's score. If it is factor it should discriminate the genes in interesting genes and the rest

TODO: it will be a good idea to replace the allGenes and allScore with an ExpressionSet class. In this way we can use tests like global test, globalAncova.... – ALL variables starting with . are just for internal class usage (private)

Objects from the Class

Objects can be created by calls of the form `new("topGOdata", ontology, allGenes, geneSelectionFun, description, annotationFun, ...)`. ~~ describe objects here ~~

Slots

`description`: Object of class `"character"` ~~

`ontology`: Object of class `"character"` ~~

`allGenes`: Object of class `"character"` ~~

`allScores`: Object of class `"ANY"` ~~

`geneSelectionFun`: Object of class `"function"` ~~

`feasible`: Object of class `"logical"` ~~

`graph`: Object of class `"graphNEL"` ~~

Methods

- **allGenes** `signature(object = "topGOdata")`: ...
- **attrInTerm** `signature(object = "topGOdata", attr = "character", whichGO = "character")`: ...
- **attrInTerm** `signature(object = "topGOdata", attr = "character", whichGO = "missing")`: ...
- **countGenesInTerm** `signature(object = "topGOdata", whichGO = "character")`: ...
- **countGenesInTerm** `signature(object = "topGOdata", whichGO = "missing")`: ...
- **description<-** `signature(object = "topGOdata")`: ...
- **description** `signature(object = "topGOdata")`: ...
- **feasible<-** `signature(object = "topGOdata")`: ...
- **feasible** `signature(object = "topGOdata")`: ...
- **geneScore** `signature(object = "topGOdata")`: ...
- **geneSelectionFun<-** `signature(object = "topGOdata")`: ...
- **geneSelectionFun** `signature(object = "topGOdata")`: ...
- **genes** `signature(object = "topGOdata")`: ...
- **genesInTerm** `signature(object = "topGOdata", whichGO = "character")`: ...
- **genesInTerm** `signature(object = "topGOdata", whichGO = "missing")`: ...
- **genTable** `signature(object = "topGOdata", resList = "list")`: ...
- **GenTable** `signature(object = "topGOdata", ...)`: ...
- **getSigGroups** `signature(object = "topGOdata", test.stat = "classicCount")`: ...
- **getSigGroups** `signature(object = "topGOdata", test.stat = "classicScore")`: ...
- **graph<-** `signature(object = "topGOdata")`: ...
- **graph** `signature(object = "topGOdata")`: ...
- **initialize** `signature(.Object = "topGOdata")`: ...
- **numGenes** `signature(object = "topGOdata")`: ...
- **ontology<-** `signature(object = "topGOdata")`: ...
- **ontology** `signature(object = "topGOdata")`: ...
- **print** `signature(x = "topGOdata")`: ...
- **sigGenes** `signature(object = "topGOdata")`: ...
- **termStat** `signature(object = "topGOdata", whichGO = "character")`: ...
- **termStat** `signature(object = "topGOdata", whichGO = "missing")`: ...
- **updateGenes** `signature(object = "topGOdata", geneList = "numeric", geneSelFun = "function")`: ...
- **updateGenes** `signature(object = "topGOdata", geneList = "factor", geneSelFun = "missing")`: ...
- **updateTerm<-** `signature(object = "topGOdata", attr = "character")`: ...
- **usedGO** `signature(object = "topGOdata")`: ...

Author(s)

Adrian Alexa

See Also

buildLevels, mapGenes2GOgraph, annFUN

Examples

```
## load the ALL dataset and the annotation library
library(ALL); data(ALL)
affyLib <- paste(annotation(ALL), "db", sep = ".")
library(package = affyLib, character.only = TRUE)

library(genefilter)
f1 <- pOverA(0.25, log2(100))
f2 <- function(x) (IQR(x) > 0.5)
ff <- filterfun(f1, f2)
ALL <- ALL[genefilter(ALL, ff), ]

## obtain the list of differentially expressed genes
## discriminate B-cell from T-cell
classLabel <- as.integer(sapply(ALL$BT, function(x) return(substr(x, 1, 1) == 'T')))

## over-expressed genes for T-cell samples
geneList <- getPvalues(exprs(ALL), classlabel = classLabel)

## the distribution of the adjusted p-values
hist(geneList, 100)
hist(geneList[geneList < 1], 100)

## define a function to select the "significant" genes
topDiffGenes <- function(allScore) {
  return(allScore < 0.01)
}

## how many differentially expressed genes are:
sum(topDiffGenes(geneList))

## build the topGOdata class
GOdata <- new("topGOdata",
              ontology = "BP",
              allGenes = geneList,
              geneSel = topDiffGenes,
              description = "GO analysis of ALL data: Differential Expression between B-c
              annot = annFUN.db,
              affyLib = affyLib)

## display the GOdata object
GOdata

##########################################################
## Examples on how to use the methods

## description of the experiment
description(GOdata)
```

```
## obtain the genes that will be used in the analysis
a <- genes(GOdata)
str(a)
numGenes(GOdata)

## obtain the score (p-value) of the genes
selGenes <- names(geneList)[sample(1:length(geneList), 10)]
gs <- geneScore(GOdata, whichGenes = selGenes)
print(gs)

## if we want an unnamed vector containing all the feasible genes
gs <- geneScore(GOdata, use.names = FALSE)
str(gs)

## the list of significant genes
sg <- sigGenes(GOdata)
str(sg)
numSigGenes(GOdata)

## to update the gene list
.geneList <- geneScore(GOdata, use.names = TRUE)
GOdata ## more available genes
GOdata <- updateGenes(GOdata, .geneList, topDiffGenes)
GOdata ## the available genes are now the feasible genes

## the available GO terms (all the nodes in the graph)
go <- usedGO(GOdata)
length(go)

## to list the genes annotated to a set of specified GO terms
sel.terms <- sample(go, 10)
ann.genes <- genesInTerm(GOdata, sel.terms)
str(ann.genes)

## the score for these genes
ann.score <- scoresInTerm(GOdata, sel.terms)
str(ann.score)

## to see the number of annotated genes
num.ann.genes <- countGenesInTerm(GOdata)
str(num.ann.genes)

## to summarise the statistics
termStat(GOdata, sel.terms)
```

`topGOresult-class` *Class "topGOresult"*

Description

Class instance created by `getSigGroups-methods`

Objects from the Class

Objects can be created by calls of the form `new("topGOresult", description, score, testName, testClass)`.

Slots

`description`: Object of class `"character"` ~~

`score`: Object of class `"numeric"` ~~

`testName`: Object of class `"character"` ~~

`testClass`: Object of class `"character"` ~~

Methods

`score`: ~~describe this method here

Author(s)

Adrian Alexa

See Also

`classicScore-class`, `groupStats-class`, `getSigGroups-methods`

`weightCount-class` *Class "weightCount"*

Description

~~ A concise (1-5 lines) description of what the class is. ~~

Details

TODO: Some details here.....

Objects from the Class

Objects can be created by calls of the form `new("weightCount", testStatistic, name, allMembers, groupMembers, sigMembers, weights, sigRatio, penalise, ...)`. ~~ describe objects here ~~

Slots

`weights`: Object of class `"numeric"` ~~

`sigRatio`: Object of class `"function"` ~~

`penalise`: Object of class `"function"` ~~

`roundFun`: Object of class `"function"` ~~

`significant`: Object of class `"integer"` ~~

`name`: Object of class `"character"` ~~

`allMembers`: Object of class `"character"` ~~

members: Object of class `"character"` ~~

testStatistic: Object of class `"function"` ~~

testStatPar: Object of class `"list"` ~~

Extends

Class `"classicCount"`, directly. Class `"groupStats"`, by class "classicCount", distance 2.

Methods

No methods defined with class "weightCount" in the signature.

Author(s)

Adrian Alexa

See Also

`classicScore-class`, `groupStats-class`, `getSigGroups-methods`

Examples

```
##---- Should be DIRECTLY executable !! ----
```

`weightScore-class` *Class "weightScore" ~~~*

Description

~~ A concise (1-5 lines) description of what the class is. ~~

Objects from the Class

Objects can be created by calls of the form `new("weightScore", testStatistic, name, allMembers, groupMembers, score, alternative, ...)`. ~~ describe objects here ~~

Slots

weights: Object of class `"numeric"` ~~

score: Object of class `"numeric"` ~~

scoreOrder: Object of class `"logical"` ~~

name: Object of class `"character"` ~~

allMembers: Object of class `"character"` ~~

members: Object of class `"character"` ~~

testStatistic: Object of class `"function"` ~~

testStatPar: Object of class `"list"` ~~

Extends

Class "classicScore", directly. Class "groupStats", by class "classicScore", distance 2.

Methods

No methods defined with class "weightScore" in the signature.

Author(s)

Adrian Alexa

See Also

classicScore-class, groupStats-class, getSigGroups-methods

Index