Package 'goSorensen'

November 9, 2026

Type Package

Title Statistical inference based on the Sorensen-Dice dissimilarity and the Gene Ontology (GO)

Version 1.14.0

Description This package implements inferential methods to compare gene lists in terms of their biological meaning as expressed in the GO. The compared gene lists are characterized by cross-tabulation frequency tables of enriched GO items. Dissimilarity between gene lists is evaluated using the Sorensen-Dice index.
The fundamental guiding principle is that two gene lists are taken as similar if they share a great proportion of common enriched GO items.

Depends R (>= 4.4)

Imports clusterProfiler, goProfiles, org.Hs.eg.db, parallel, stats, stringr

Suggests BiocManager, BiocStyle, knitr, rmarkdown, org.At.tair.db, org.Ag.eg.db, org.Bt.eg.db, org.Ce.eg.db, org.Cf.eg.db, org.Dm.eg.db, org.Dr.eg.db, org.EcSakai.eg.db, org.EcK12.eg.db, org.Gg.eg.db, org.Mm.eg.db, org.Mmu.eg.db, org.Rn.eg.db, org.Sc.sgd.db, org.Ss.eg.db, org.Pt.eg.db, org.Xl.eg.db, GO.db, ggplot2, ggrepel, DT, magick

VignetteBuilder knitr

License GPL-3

Encoding UTF-8

LazyData false

RoxygenNote 7.3.3

NeedsCompilation no

biocViews Annotation, GO, GeneSetEnrichment, Software, Microarray, Pathways, GeneExpression, MultipleComparison, GraphAndNetwork, Reactome, Clustering, KEGG

Author Pablo Flores [aut, cre] (<https://orcid.org/0000-0002-7156-8547>), Jordi Ocana [aut, ctb] (0000-0002-4736-699), Alexandre Sanchez-Pla [ctb] (<https://orcid.org/0000-0002-8673-7737>), Miquel Salicru [ctb] (<https://orcid.org/0000-0001-9644-5626>)

Maintainer Pablo Flores <p_flores@espoch.edu.ec>

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

git_branch RELEASE_3_23

git_last_commit 04e9f1e

git_last_commit_date 2026-04-28

Repository Bioconductor 3.23

Date/Publication 2026-11-08

Contents

allBuildEnrichTable . . . 3
allContTabs . . . 5
allDissMatrx . . . 5
allEqTests . . . 6
allEqTests_boot . . . 6
allEquivTestSorensen . . . 7
allHclustThreshold . . . 9
allOncoGeneLists . . . 10
allSorenThreshold . . . 11
boot.tStat . . . 13
buildEnrichTable . . . 14
cont_all_BP4 . . . 17
cont_atlas.sanger_BP4 . . . 18
dissMatrx_BP4 . . . 19
dSorensen . . . 19
duppSorensen . . . 22
enrichedIn . . . 27
enrichedInBP4 . . . 30
eqTest_all_BP4 . . . 31
eqTest_atlas.sanger_BP4 . . . 32
equivTestSorensen . . . 32
fullEnrichedInBP4 . . . 37
getDissimilarity . . . 38
getEffNboot . . . 40
getNboot . . . 43
getPvalue . . . 46
getSE . . . 48
getTable . . . 50
getUpper . . . 52
gosorensen . . . 55
hclustThreshold . . . 56
nice2x2Table . . . 58
pbtGeneLists . . . 59
pruneClusts . . . 60
seSorensen . . . 60
sorenThreshold . . . 63

upgrade 66

Index **68**

`allBuildEnrichTable` *Iterate* `buildEnrichTable` *along the specified GO ontologies and GO levels*

Description

Iterate `buildEnrichTable` along the specified GO ontologies and GO levels

Usage

```
allBuildEnrichTable(
  x,
  check.table = TRUE,
  geneUniverse,
  orgPackg,
  ontos = c("BP", "CC", "MF"),
  GOLevels = seq.int(3, 10),
  keyType = "ENTREZID",
  storeEnrichedIn = TRUE,
  trace = TRUE,
  ...
)
```

Arguments

`x`	object of class "list". Each of its elements must be a "character" vector of gene identifiers (e.g., ENTREZ). Then all pairwise contingency tables of joint enrichment are built between these gene lists, iterating the process for all specified GO ontologies and GO levels.
`check.table`	Boolean. If TRUE (default), all resulting tables are checked by means of function `nice2x2Table`.
`geneUniverse`	character vector containing the universe of genes from where gene lists have been extracted. This vector must be obtained from the annotation package declared in `orgPackg`. For more details, refer to vignette goSorensen_Introduction.
`orgPackg`	A string with the name of the genomic annotation package corresponding to a specific species to be analysed, which must be previously installed and activated. For more details, refer to vignette goSorensen_Introduction.
`ontos`	"character", GO ontologies to analyse. Defaults to `c("BP", "CC", "MF")`.
`GOLevels`	"integer", GO levels to analyse inside each of these GO ontologies.
`keyType`	character keytype of input gene, e.g., ENTREZID or SYMBOL
`storeEnrichedIn`	logical, for each ontology and level under study, the matrix of enriched (GO terms) x (gene lists) TRUE/FALSE values, must be stored in the result?

trace	Logical. If TRUE (default), the (usually very time consuming) process of function allbuildEnrichTable is traced along the specified GO ontologies and levels.
...	extra parameters for function buildEnrichTable.

Value

An object of class "allTableList". It is a list with as many components as GO ontologies have been analysed. Each of these elements is itself a list with as many components as GO levels have been analised. Finally, the elements of these lists are objects as generated by buildEnrichTable.list, i.e., objects of class "tableList" containing all pairwise contingency tables of mutual enrichment between the gene lists in argument x.

Examples

```
# This example is highly time-consuming. It scans two GO ontologies and three
# GO levels inside them to obtain the contingency tables of joint enrichment.

# Obtaining ENTREZ identifiers for the gene universe of humans:
# library(org.Hs.eg.db)
# humanEntrezIDs <- keys(org.Hs.eg.db, keytype = "ENTREZID")

# Gene lists to be explored for enrichment:
# data(allOncoGeneLists)

# Computing Contingency Tables for all the possible pairwise comparisons for
# the ontologies MF, BP, and the GO levels from 4 to 6:
# someOntosAndLevels <- allBuildEnrichTable(allOncoGeneLists,
#                                           geneUniverse = humanEntrezIDs,
#                                           orgPackg = "org.Hs.eg.db",
#                                           ontos = c("MF", "BP"),
#                                           GOLevels = seq.int(4,6))
# someOntosAndLevels$BP$`level 4`
# attr(someOntosAndLevels$BP$`level 4`, "enriched")
#
# To avoid storage-consuming redundancies, the table of GO terms x gene lists
# enrichment is not stored for the full set of gene list pairs at each
# ontology and level
# someOntosAndLevels$BP$`level 4`$Vogelstein
# attr(someOntosAndLevels$BP$`level 4`$Vogelstein, "enriched")
# someOntosAndLevels$BP$`level 4`$Vogelstein$atlas
# attr(someOntosAndLevels$BP$`level 4`$Vogelstein$atlas, "enriched")
#
# When the "ontos" and/or "GOLevels" arguments are not supplied, the function
# computes by default every possible contingency table between the lists
# being compared for the three ontologies (BP, CC, MF) and/or GO levels from
# 3 to 10.
```

allContTabs *Example of the output produced by the function* allBuildEnrichTable.

Description

This object contains all the enrichment contingency tables to compare all possible pairs of lists from allOncoGeneLists across GO-Levels 3 to 10, and for the ontologies BP, CC, and MF.

Usage

```
data(allContTabs)
```

Format

An exclusive object from goSorensen of the class "allTableList"

Details

The attribute enriched is present in each element of this output, meaning that there is an enrichment matrix, similar to the one obtained with the function enrichedIn, for each ontology and GO-Level contained in this object.

Consider this object only as an illustrative example, which is valid exclusively for the data allOncoGeneLists contained in this package. Note that gene lists, GO terms, and Bioconductor may change over time. The current version of these results were generated with Bioconductor version 3.20.

allDissMatrx *Example of the output produced by the function* allSorenThreshold. *It contains the dissimilarity matrices for GO levels from 3 to 10 across the ontologies BP, CC and MF.*

Description

This object contains the matrices of dissimilarities between the 7 lists from allOncoGeneLists, computed based on the irrelevance threshold that makes them equivalent for GO levels from 3 to 10 across the ontologies BP, CC and MF.

Usage

```
data("allDissMatrx")
```

Format

An object of class "dist"

Details

Equivalence tests were computed based on the normal distribution (`boot = TRUE` by default) and using a confidence level `conf.level = 0.95`.

Consider this object only as an illustrative example, which is valid exclusively for the data `allOncoGeneLists` contained in this package. Note that gene lists, GO terms, and Bioconductor may change over time. The current version of these results were generated with Bioconductor version 3.20.

`allEqTests` *Example of the output produced by the function* `allEquivTestSorensen` *using the normal asymptotic distribution.*

Description

This object contains all the outputs for the equivalence tests to compare all possible pairs of lists from `allOncoGeneLists` across GO-Levels 3 to 10, and for the ontologies BP, CC, and MF, using the normal asymptotic distribution.

Usage

```
data(allEqTests)
```

Format

An exclusive object from `goSorensen` of the class "AllEquivSDhtest"

Details

The parameters considered to execute these tests are: irrelevance limit `d0 = 0.4444` and confidence level `conf.level = 0.95`.

Consider this object only as an illustrative example, which is valid exclusively for the data `allOncoGeneLists` contained in this package. Note that gene lists, GO terms, and Bioconductor may change over time. The current version of these results were generated with Bioconductor version 3.20.

`allEqTests_boot` *Example of the output produced by the function* `allEquivTestSorensen` *using the approximated bootstrap distribution.*

Description

This object contains all the outputs for the equivalence tests to compare all possible pairs of lists from `allOncoGeneLists` across GO-Levels 3 to 10, and for the ontologies BP, CC, and MF, using the approximated bootstrap distribution.

Usage

```
data(allEqTests_boot)
```

Format

An exclusive object from goSorensen of the class "AllEquivSDhtest"

Details

The parameters considered to execute these tests are: irrelevance limit d0 = 0.4444 and confidence level conf.level = 0.95.

Consider this object only as an illustrative example, which is valid exclusively for the data allOncoGeneLists contained in this package. Note that gene lists, GO terms, and Bioconductor may change over time. The current version of these results were generated with Bioconductor version 3.20.

---

allEquivTestSorensen *Iterate* equivTestSorensen *along the specified GO ontologies and GO levels*

Description

Iterate equivTestSorensen along the specified GO ontologies and GO levels

Usage

```
allEquivTestSorensen(x, ...)

## S3 method for class 'list'
allEquivTestSorensen(
  x,
  d0 = 1/(1 + 1.25),
  conf.level = 0.95,
  boot = FALSE,
  nboot = 10000,
  check.table = TRUE,
  ontos = c("BP", "CC", "MF"),
  GOLevels = seq.int(3, 10),
  trace = TRUE,
  ...
)

## S3 method for class 'allTableList'
allEquivTestSorensen(
  x,
  d0 = 1/(1 + 1.25),
  conf.level = 0.95,
  boot = FALSE,
```

```
  nboot = 10000,
  check.table = TRUE,
  ontos,
  GOLevels,
  trace = TRUE,
  ...
)
```

Arguments

x	either an object of class "list" or an object of class "allTableList". In the first case, each of its elements must be a "character" vector of gene identifiers (e.g., ENTREZ).
...	extra parameters for function buildEnrichTable.
d0	equivalence threshold for the Sorensen-Dice dissimilarity, d. The null hypothesis states that d >= d0, i.e., inequivalence between the compared gene lists and the alternative that d < d0, i.e., equivalence or dissimilarity irrelevance (up to a level d0).
conf.level	confidence level of the one-sided confidence interval, a value between 0 and 1.
boot	boolean. If TRUE, the confidence interval and the test p-value are computed by means of a bootstrap approach instead of the asymptotic normal approach. Defaults to FALSE.
nboot	numeric, number of initially planned bootstrap replicates. Ignored if boot == FALSE. Defaults to 10000.
check.table	Boolean. If TRUE (default), argument x is checked to adequately represent a 2x2 contingency table (or an aggregate of them) or gene lists producing a correct table. This checking is performed by means of function nice2x2Table.
ontos	"character", GO ontologies to analyse. Defaults to c("BP", "CC", "MF").
GOLevels	"integer", GO levels to analyse inside each one of the GO ontologies.
trace	Logical. If TRUE (default), the (usually very time consuming) process of function allEquivTestSorensen is traced along the specified GO ontologies and levels.

Value

An object of class "AllEquivSDhtest". It is a list with as many components as GO ontologies have been analysed. Each of these elements is itself a list with as many components as GO levels have been analized. Finally, the elements of these lists are objects as generated by equivTestSorensen.list, i.e., objects of class "equivSDhtestList" containing pairwise comparisons between gene lists.

Methods (by class)

- allEquivTestSorensen(list): S3 method for class "list"
- allEquivTestSorensen(allTableList): S3 method for class "allTableList"

Examples

```
# Gene lists to be explored for enrichment:
data(allOncoGeneLists)

# Obtaining ENTREZ identifiers for the gene universe of humans:
library(org.Hs.eg.db)
humanEntrezIDs <- keys(org.Hs.eg.db, keytype = "ENTREZID")

# This example is highly time-consuming. It scans two GO ontologies and three
# GO levels inside them to perform the equivalence test.
# allEquivTestSorensen(allOncoGeneLists,
#                      geneUniverse = humanEntrezIDs, orgPackg = "org.Hs.eg.db",
#                      ontos = c("MF", "BP"), GOLevels = seq.int(4,6))
# When the "ontos" and "GOLevels" arguments are not supplied, the function computes
# by default every possible contingency table between the lists being compared for
# the three ontologies (BP, CC, MF) and GO levels from 3 to 10.
#
# Much faster:
# Object \code{allContTabs} of class "allTableList" contains all the pairwise contingency tables of
# joint enrichment for the gene lists in \code{allOncoGeneLists}, obtained along all three GO
# ontologies and along GO levels 3 to 10:
data(allContTabs)
tests <- allEquivTestSorensen(allContTabs, ontos = c("MF", "BP"), GOLevels = seq.int(4,6))
tests$BP$`level 5`
getPvalue(tests)
```

---

allHclustThreshold *Iterate* hclustThreshold *along the specified GO ontologies and GO levels*

Description

Iterate hclustThreshold along the specified GO ontologies and GO levels

Usage

```
allHclustThreshold(x, ontos, GOLevels, trace = TRUE, ...)
```

Arguments

x	an object of class "distList".
ontos	"character", GO ontologies to iterate. Defaults to the ontologies in 'x'.
GOLevels	"integer", GO levels to iterate inside each one of these GO ontologies.
trace	Logical. If TRUE (default), the process is traced along the specified GO ontologies and levels.
...	extra parameters for function hclustThreshold.

Value

An object of class "equivClustSorensenList" descending from "iterEquivClust" which itself descends from class "list". It is a list with as many components as GO ontologies have been specified. Each of these elements is itself a list with as many components as GO levels have been specified. Finally, the elements of these lists are objects of class "equivClustSorensen", descending from "equivClust" which itself descends from "hclust".

Examples

```
# Object \code{allTabs} of class "allTableList" contains all the pairwise contingency tables of
# joint enrichment for the gene lists in \code{allOncoGeneLists}, obtained along all three GO
# ontologies and along GO levels 3 to 10:
data(allContTabs)
# Compute the Sorensen-Dice equivalence threshold dissimilarity (only for the MF and CC
# ontologies and from levels 4 to 6):
dists <- allSorenThreshold(allContTabs, ontos = c("MF", "CC"), GOLevels = seq.int(4,6))
hclusts <- allHclustThreshold(dists)
hclusts$MF$`level 6`
plot(hclusts$MF$`level 6`)
```

---

allOncoGeneLists *7 gene lists possibly related with cancer*

Description

An object of class "list" of length 7. Each one of its elements is a "character" vector of gene identifiers (e.g., ENTREZ). Only gene lists of length almost 100 were taken from their source web. Take these lists just as an illustrative example, they are not automatically updated.

Usage

```
data(allOncoGeneLists)
```

Format

An object of class "list" of length 7. Each one of its elements is a "character" vector of ENTREZ gene identifiers .

Source

http://www.bushmanlab.org/links/genelists

`allSorenThreshold` *Iterate* `sorenThreshold` *along the specified GO ontologies and GO levels*

Description

Iterate `sorenThreshold` along the specified GO ontologies and GO levels

Usage

```
allSorenThreshold(x, ...)

## S3 method for class 'list'
allSorenThreshold(
  x,
  geneUniverse,
  orgPackg,
  boot = FALSE,
  nboot = 10000,
  boot.seed = 6551,
  ontos = c("BP", "CC", "MF"),
  GOLevels = seq.int(3, 10),
  trace = TRUE,
  alpha = 0.05,
  precis = 0.001,
  ...
)

## S3 method for class 'allTableList'
allSorenThreshold(
  x,
  boot = FALSE,
  nboot = 10000,
  boot.seed = 6551,
  ontos,
  GOLevels,
  trace = TRUE,
  alpha = 0.05,
  precis = 0.001,
  ...
)
```

Arguments

`x`	either an object of class "list" or an object of class "allTableList". In the first case, each of its elements must be a "character" vector of gene identifiers (e.g., ENTREZ). In the second case, the object corresponds to all contingency tables of joint enrichment along one or more GO ontologies and one or more GO levels.

...	extra parameters for function buildEnrichTable.
geneUniverse	character vector containing the universe of genes from where gene lists have been extracted. This vector must be obtained from the annotation package declared in orgPackg. For more details see README File.
orgPackg	A string with the name of the genomic annotation package corresponding to a specific species to be analyzed, which must be previously installed and activated. For more details see README File.
boot	boolean. If TRUE, the confidence intervals and the test p-values are computed by means of a bootstrap approach instead of the asymptotic normal approach. Defaults to FALSE.
nboot	numeric, number of initially planned bootstrap replicates. Ignored if boot == FALSE. Defaults to 10000.
boot.seed	starting random seed for all bootstrap iterations. Defaults to 6551. see the details section
ontos	"character", GO ontologies to analyse.
GOLevels	"integer", GO levels to analyse inside each one of these GO ontologies.
trace	Logical. If TRUE (default), the (usually very time consuming) process is traced along the specified GO ontologies and levels.
alpha	simultaneous nominal significance level for the equivalence tests to be repeteadly performed, defaults to 0.05
precis	numerical precision in the iterative search of the equivalence threshold dissimilarities,

Value

An object of class "distList". It is a list with as many components as GO ontologies have been analysed. Each of these elements is itself a list with as many components as GO levels have been analysed. Finally, the elements of these lists are objects of class "dist" with the Sorensen-Dice equivalence threshold dissimilarity.

Methods (by class)

- allSorenThreshold(list): S3 method for class "list"
- allSorenThreshold(allTableList): S3 method for class "allTableList"

Examples

```
# # This example is highly time-consuming. It scans two GO ontologies and three
# # GO levels inside them to perform the equivalence test.

# Obtaining ENTREZ identifiers for the gene universe of humans:
# library(org.Hs.eg.db)
# humanEntrezIDs <- keys(org.Hs.eg.db, keytype = "ENTREZID")

# # Gene lists to be explored for enrichment:
# data("allOncoGeneLists")
# allSorenThreshold(allOncoGeneLists,
```

```
#                   geneUniverse = humanEntrezIDs, orgPackg = "org.Hs.eg.db",
#
# Much faster:
# Object allContTabs of class "allTableList" contains all the pairwise contingency tables of
# joint enrichment for the gene lists in \code{allOncoGeneLists}, obtained along all three GO
# ontologies and along GO levels 3 to 10:
data(allContTabs)
dSors <- allSorenThreshold(allContTabs, ontos = c("MF", "BP"), GOLevels = seq.int(4,6))
dSors$BP$`level 5`
```

`boot.tStat` *Studentized Sorensen-Dice dissimilarity statistic*

Description

Efficient computation of the studentized statistic (^dis - dis) / ^se where 'dis' stands for the "population" value of the Sorensen-Dice dissimilarity, '^dis' for its estimated value and '^se'for the estimate of the standard error of '^dis'. Internally used in bootstrap computations.

Usage

```
boot.tStat(xBoot, dis)
```

Arguments

`xBoot`	either an object of class "table", "matrix" or "numeric" representing a 2x2 contingency table of joint enrichment.
`dis`	the "known" value of the population dissimilarity.

Details

This function is repeatedly evaluated during bootstrap iterations. Given a contingency table 'x' of mutual enrichment (the "true" dataset):

$$\begin{matrix} n_{11} & n_{10} \\ n_{01} & n_{00}, \end{matrix}$$

summarizing the status of mutual presence of enrichment in two gene lists, where the subindex '11' corresponds to those GO terms enriched in both lists, '01' to terms enriched in the second list but not in the first one, '10' to terms enriched in the first list but not enriched in the second one and '00' to those GO terms non enriched in both gene lists, i.e., to the double negatives.

A typical bootstrap iteration consists in repeatedly generating four frequencies from a multinomial of parameters size = sum(n_ij), i,j = 1, 0 and probabilities (n_11/size, n_10/size, n_10/size, n_00/size). The argument 'xBoot' corresponds to each one of these bootstrap resamples (indiferenly represented in form of a 2x2 "table" or "matrix" or as a numeric vector) In each bootstrap iteration, the value of the "true" known 'dis' is the dissimilarity which was computed from 'x' (a constant, known value in the full iteration) and the values of '^dis' and '^se' are internally computed from the bootstrap data 'xBoot'.

Value

A numeric value, the result of computing (^dis - dis) / ^se.

`buildEnrichTable` *Creates a 2x2 enrichment contingency table from two gene lists, or all pairwise contingency tables for a "list" of gene lists.*

Description

Creates a 2x2 enrichment contingency table from two gene lists, or all pairwise contingency tables for a "list" of gene lists.

Usage

```
buildEnrichTable(x, ...)

## Default S3 method:
buildEnrichTable(
  x,
  y,
  listNames = c("gene.list1", "gene.list2"),
  check.table = TRUE,
  geneUniverse,
  orgPackg,
  onto,
  GOLevel,
  storeEnrichedIn = TRUE,
  pAdjustMeth = "BH",
  pvalCutoff = 0.01,
  qvalCutoff = 0.05,
  keyType = "ENTREZID",
  parallel = FALSE,
  nOfCores = 1,
  ...
)

## S3 method for class 'character'
buildEnrichTable(
  x,
  y,
  listNames = c("gene.list1", "gene.list2"),
  check.table = TRUE,
  geneUniverse,
  orgPackg,
  onto,
  GOLevel,
```

```
  storeEnrichedIn = TRUE,
  pAdjustMeth = "BH",
  pvalCutoff = 0.01,
  qvalCutoff = 0.05,
  keyType = "ENTREZID",
  parallel = FALSE,
  nOfCores = min(detectCores() - 1),
  ...
)

## S3 method for class 'list'
buildEnrichTable(
  x,
  check.table = TRUE,
  geneUniverse,
  orgPackg,
  onto,
  GOLevel,
  storeEnrichedIn = TRUE,
  pAdjustMeth = "BH",
  pvalCutoff = 0.01,
  qvalCutoff = 0.05,
  keyType = "ENTREZID",
  parallel = FALSE,
  nOfCores = min(detectCores() - 1, length(x) - 1),
  ...
)
```

Arguments

x	either an object of class "character" (or coerzable to "character") representing a vector of gene identifiers (e.g., ENTREZ) or an object of class "list". In this second case, each element of the list must be a "character" vector of gene identifiers (e.g., ENTREZ). Then, all pairwise contingency tables between these gene lists are built.
...	Additional parameters for internal use (not used for the moment)
y	an object of class "character" (or coerzable to "character") representing a vector of gene identifiers (e.g., ENTREZ).
listNames	a character(2) with the gene lists names originating the cross-tabulated enrichment frequencies. Only in the "character" or default interface.
check.table	Logical The resulting table must be checked. Defaults to TRUE.
geneUniverse	character vector containing the universe of genes from where gene lists have been extracted. This vector must be obtained from the annotation package declared in orgPackg. For more details, refer to vignette goSorensen_Introduction.
orgPackg	A string with the name of the genomic annotation package corresponding to a specific species to be analysed, which must be previously installed and activated. For more details, refer to vignette goSorensen_Introduction.

onto	string describing the ontology. Either "BP", "MF" or "CC".
GOLevel	An integer, the GO ontology level.
storeEnrichedIn	logical, the matrix of enriched (GO terms) x (gene lists) TRUE/FALSE values, must be stored in the result? See the details section
pAdjustMeth	string describing the adjust method, either "BH", "BY" or "Bonf", defaults to 'BH'.
pvalCutoff	adjusted pvalue cutoff on enrichment tests to report
qvalCutoff	qvalue cutoff on enrichment tests to report as significant. Tests must pass i) pvalueCutoff on unadjusted pvalues, ii) pvalueCutoff on adjusted pvalues and iii) qvalueCutoff on qvalues to be reported
keyType	character keytype of input gene, e.g., ENTREZID or SYMBOL
parallel	Logical. Defaults to FALSE but put it at TRUE for parallel computation.
nOfCores	Number of cores for parallel computations. Only in "list" interface.

Details

If the argument storeEnrichedIn is TRUE (the default value), the result of buildEnrichTable includes an additional attribute enriched with a matrix of TRUE/FALSE values. Each of its rows indicates if a given GO term is enriched or not in each one of the gene lists (columns). To save space, only GO terms enriched in almost one of the gene lists are included in this matrix.

Also to avoid redundancies and to save space, the result of buildEnrichTable.list (an object of class "tableList", which is itself an aggregate of 2x2 contingency tables of class "table") has the attribute enriched but its table members do not have this attribute.

The default value of argument parallel ís "FALSE" and you may consider the trade of between the time spent in initializing parallelization and the possible time gain when parallelizing. It is difficult to establish a general guideline, but parallelizing is only worthwhile when analyzing many gene lists, on the order of 30 or more, although it depends on each computer and each application.

Value

in the "character" interface, an object of class "table". It represents a 2x2 contingency table, the cross-tabulation of the enriched GO terms in two gene lists: "Number of enriched GO terms in list 1 (TRUE, FALSE)" x "Number of enriched Go terms in list 2 (TRUE, FALSE)". In the "list" interface, the result is an object of class "tableList" with all pairwise tables. Class "tableList" corresponds to objects representing all mutual enrichment contingency tables generated in a pairwise fashion: Given gene lists (i.e. "character" vectors of gene identifiers) l1, l2, ..., lk, an object of class "tableList" is a list of lists of contingency tables t(i,j) generated from each pair of gene lists i and j, with the following structure:

$l2

$l2$l1$t(2,1)

$l3

$l3$l1$t(3,1), $l3$l2$t(3,2)

...

$lk

lkl1$t(k,1), lkl2$t(k,2), ..., lkl(k-1)t(K,k-1)

Methods (by class)

- buildEnrichTable(default): S3 default method
- buildEnrichTable(character): S3 method for class "character"
- buildEnrichTable(list): S3 method for class "list"

Examples

```
# Obtaining ENTREZ identifiers for the gene universe of humans:
library(org.Hs.eg.db)
humanEntrezIDs <- keys(org.Hs.eg.db, keytype = "ENTREZID")

# Gene lists to be explored for enrichment:
data(allOncoGeneLists)
?allOncoGeneLists

# Table of joint GO term enrichment between gene lists Vogelstein and sanger,
# for ontology MF at GO level 6.
vog.VS.sang <- buildEnrichTable(allOncoGeneLists[["Vogelstein"]],
                                allOncoGeneLists[["sanger"]],
                               geneUniverse = humanEntrezIDs, orgPackg = "org.Hs.eg.db",
                          onto = "MF", GOLevel = 6, listNames = c("Vogelstein", "sanger"))
vog.VS.sang
attr(vog.VS.sang, "enriched")

# All tables of mutual enrichment:
all.tabs <- buildEnrichTable(allOncoGeneLists,
                             geneUniverse = humanEntrezIDs, orgPackg = "org.Hs.eg.db",
                             onto = "MF", GOLevel = 6)
attr(all.tabs, "enriched")
all.tabs$waldman
all.tabs$waldman$atlas
attr(all.tabs$waldman$atlas, "enriched")
```

---

cont_all_BP4 *Example of the output produced by the function* buildEnrichTable. *It contains the enrichment contingency tables for all the lists from* allOncoGeneLists *at level 4 of ontology BP.*

Description

Given 7 lists contained in allOncoGeneLists, this object contains the 7(6)/2 = 21 possible enrichment contingency tables to compare all possible pairs of lists. Each contingency 2x2 table contains the number of joint enriched GO terms (TRUE-TRUE); the number of GO terms enriched only in one list but not in the other one (FALSE-TRUE and TRUE-FALSE); and the number of GO terms not enriched in either of the two lists.

An important attribute of this object is enriched, which contains the enrichment matrix obtained using the function enrichedIn. Actually, the contingency tables in this object are derived from cross-frequency tables created between pairs of lists, which are located as columns in this enrichment matrix.

Usage

```
data(cont_all_BP4)
```

Format

An exclusive object from goSorensen of the class "tableList"

Details

Consider this object only as an illustrative example, which is valid exclusively for the data allOncoGeneLists contained in this package. Note that gene lists, GO terms, and Bioconductor may change over time. The current version of these results were generated with Bioconductor version 3.20.

---

cont_atlas.sanger_BP4 *Example of the output produced by the function* buildEnrichTable. *It contains the enrichment contingency table for two lists at level 4 of ontology BP.*

Description

A contingency 2x2 table with the number of joint enriched GO terms (TRUE-TRUE); the number of GO terms enriched only in one list but not in the other one (FALSE-TRUE and TRUE-FALSE); and the number of GO terms not enriched in either of the two lists.

Usage

```
data(cont_atlas.sanger_BP4)
```

Format

An object of class "table"

Details

Consider this object only as an illustrative example, which is valid exclusively for the lists atlas and sanger from the data allOncoGeneLists contained in this package. Note that gene lists, GO terms, and Bioconductor may change over time. The current version of these results were generated with Bioconductor version 3.20.

dissMatrx_BP4 *Example of the output produced by the function* sorenThreshold. *It contains the dissimilarity matrix at GO level 4, for the ontology BP.*

Description

This object contains the matrix of dissimilarities between the 7 lists from allOncoGeneLists, computed based on the irrelevance threshold that makes them equivalent at GO level 4, for the ontology BP.

Usage

```
data("dissMatrx_BP4")
```

Format

An object of class "dist"

Details

Equivalence tests were computed based on the normal distribution (boot = TRUE by default) and using a confidence level conf.level = 0.95.

Consider this object only as an illustrative example, which is valid exclusively for the data allOncoGeneLists contained in this package. Note that gene lists, GO terms, and Bioconductor may change over time. The current version of these results were generated with Bioconductor version 3.20.

dSorensen *Computation of the Sorensen-Dice dissimilarity*

Description

Computation of the Sorensen-Dice dissimilarity

Usage

```
dSorensen(x, ...)

## S3 method for class 'table'
dSorensen(x, check.table = TRUE, ...)

## S3 method for class 'matrix'
dSorensen(x, check.table = TRUE, ...)

## S3 method for class 'numeric'
dSorensen(x, check.table = TRUE, ...)
```

```
## S3 method for class 'character'
dSorensen(x, y, check.table = TRUE, ...)

## S3 method for class 'list'
dSorensen(x, check.table = TRUE, ...)

## S3 method for class 'tableList'
dSorensen(x, check.table = TRUE, ...)
```

Arguments

x	either an object of class "table", "matrix" or "numeric" representing a 2x2 contingency table, or a "character" vector (a set of gene identifiers) or "list" or "tableList" object. See the details section for more information.
...	extra parameters for function buildEnrichTable.
check.table	Boolean. If TRUE (default), argument x is checked to adequately represent a 2x2 contingency table, by means of function nice2x2Table.
y	an object of class "character" representing a vector of valid gene identifiers (e.g., ENTREZ).

Details

Given a 2x2 arrangement of frequencies (either implemented as a "table", a "matrix" or a "numeric" object):

$$\begin{matrix} n_{11} & n_{10} \\ n_{01} & n_{00}, \end{matrix}$$

this function computes the Sorensen-Dice dissimilarity

$$\frac{n_{10} + n_{01}}{2n_{11} + n_{10} + n_{01}}.$$

The subindex '11' corresponds to those GO terms enriched in both lists, '01' to terms enriched in the second list but not in the first one, '10' to terms enriched in the first list but not enriched in the second one and '00' corresponds to those GO terms non enriched in both gene lists, i.e., to the double negatives, a value which is ignored in the computations.

In the "numeric" interface, if length(x) >= 3, the values are interpreted as $(n_{11}, n_{01}, n_{10}, n_{00})$, always in this order and discarding extra values if necessary. The result is correct, regardless the frequencies being absolute or relative.

If x is an object of class "character", then x (and y) must represent two "character" vectors of valid gene identifiers (e.g., ENTREZ). Then the dissimilarity between lists x and y is computed, after internally summarizing them as a 2x2 contingency table of joint enrichment. This last operation is performed by function buildEnrichTable and "valid gene identifiers (e.g., ENTREZ)" stands for the coherency of these gene identifiers with the arguments geneUniverse and orgPackg of buildEnrichTable, passed by the ellipsis argument ... in dSorensen.

If x is an object of class "list", the argument must be a list of "character" vectors, each one representing a gene list (character identifiers). Then, all pairwise dissimilarities between these gene lists are computed.

If x is an object of class "tableList", the Sorensen-Dice dissimilarity is computed over each one of these tables. Given k gene lists (i.e. "character" vectors of gene identifiers) l1, l2, ..., lk, an object of class "tableList" (typically constructed by a call to function buildEnrichTable) is a list of lists of contingency tables t(i,j) generated from each pair of gene lists i and j, with the following structure:

$l2

$l2$l1$t(2,1)

$l3

$l3$l1$t(3,1), $l3$l2$t(3,2)

...

$lk

lkl1$t(k,1), lkl2$t(k,2), ..., lkl(k-1)t(k,k-1)

Value

In the "table", "matrix", "numeric" and "character" interfaces, the value of the Sorensen-Dice dissimilarity. In the "list" and "tableList" interfaces, the symmetric matrix of all pairwise Sorensen-Dice dissimilarities.

Methods (by class)

- dSorensen(table): S3 method for class "table"
- dSorensen(matrix): S3 method for class "matrix"
- dSorensen(numeric): S3 method for class "numeric"
- dSorensen(character): S3 method for class "character"
- dSorensen(list): S3 method for class "list"
- dSorensen(tableList): S3 method for class "tableList"

See Also

buildEnrichTable for constructing contingency tables of mutual enrichment, nice2x2Table for checking contingency tables validity, seSorensen for computing the standard error of the dissimilarity, duppSorensen for the upper limit of a one-sided confidence interval of the dissimilarity, equivTestSorensen for an equivalence test.

Examples

```
# Gene lists 'atlas' and 'sanger' in 'allOncoGeneLists' dataset. Table of joint enrichment
# of GO terms in ontology BP at level 3.
data(cont_atlas.sanger_BP4)
cont_atlas.sanger_BP4
?cont_atlas.sanger_BP4
dSorensen(cont_atlas.sanger_BP4)
```

```
# Table represented as a vector:
conti4 <- c(56, 1, 30, 471)
dSorensen(conti4)
# or as a plain matrix:
dSorensen(matrix(conti4, nrow = 2))

# This function is also appropriate for proportions:
dSorensen(conti4 / sum(conti4))

conti3 <- c(56, 1, 30)
dSorensen(conti3)

# Sorensen-Dice dissimilarity from scratch, directly from two gene lists:
# (These examples may be considerably time consuming due to many enrichment
# tests to build the contingency tables of joint enrichment)
# data(allOncoGeneLists)
# ?allOncoGeneLists

# Obtaining ENTREZ identifiers for the gene universe of humans:
# library(org.Hs.eg.db)
# humanEntrezIDs <- keys(org.Hs.eg.db, keytype = "ENTREZID")

# (Time consuming, building the table requires many enrichment tests:)
# dSorensen(allOncoGeneLists$atlas, allOncoGeneLists$sanger,
#           onto = "BP", GOLevel = 3,
#           geneUniverse = humanEntrezIDs, orgPackg = "org.Hs.eg.db")

# Essentially, the above code makes the same as:
# cont_atlas.sanger_BP4 <- buildEnrichTable(allOncoGeneLists$atlas, allOncoGeneLists$sanger,
#                                     onto = "BP", GOLevel = 4,
#                                 geneUniverse = humanEntrezIDs, orgPackg = "org.Hs.eg.db")
# dSorensen(cont_atlas.sanger_BP4)
# (Quite time consuming, all pairwise dissimilarities:)
# dSorensen(allOncoGeneLists,
#           onto = "BP", GOLevel = 4,
#           geneUniverse = humanEntrezIDs, orgPackg = "org.Hs.eg.db")
```

---

duppSorensen *Upper limit of a one-sided confidence interval (0, dUpp] for the Sorensen-Dice dissimilarity*

Description

Upper limit of a one-sided confidence interval (0, dUpp] for the Sorensen-Dice dissimilarity

Usage

```
duppSorensen(x, ...)

## S3 method for class 'table'
```

```
duppSorensen(
  x,
  dis = dSorensen.table(x, check.table = FALSE),
  se = seSorensen.table(x, check.table = FALSE),
  conf.level = 0.95,
  z.conf.level = qnorm(1 - conf.level),
  boot = FALSE,
  nboot = 10000,
  check.table = TRUE,
  ...
)

## S3 method for class 'matrix'
duppSorensen(
  x,
  dis = dSorensen.matrix(x, check.table = FALSE),
  se = seSorensen.matrix(x, check.table = FALSE),
  conf.level = 0.95,
  z.conf.level = qnorm(1 - conf.level),
  boot = FALSE,
  nboot = 10000,
  check.table = TRUE,
  ...
)

## S3 method for class 'numeric'
duppSorensen(
  x,
  dis = dSorensen.numeric(x, check.table = FALSE),
  se = seSorensen.numeric(x, check.table = FALSE),
  conf.level = 0.95,
  z.conf.level = qnorm(1 - conf.level),
  boot = FALSE,
  nboot = 10000,
  check.table = TRUE,
  ...
)

## S3 method for class 'character'
duppSorensen(
  x,
  y,
  conf.level = 0.95,
  boot = FALSE,
  nboot = 10000,
  check.table = TRUE,
  ...
)
```

```
## S3 method for class 'list'
duppSorensen(
  x,
  conf.level = 0.95,
  boot = FALSE,
  nboot = 10000,
  check.table = TRUE,
  ...
)

## S3 method for class 'tableList'
duppSorensen(
  x,
  conf.level = 0.95,
  boot = FALSE,
  nboot = 10000,
  check.table = TRUE,
  ...
)
```

Arguments

x	either an object of class "table", "matrix" or "numeric" representing a 2x2 contingency table, or a "character" (a set of gene identifiers) or "list" or "tableList" object. See the details section for more information.
...	additional arguments for function buildEnrichTable.
dis	Sorensen-Dice dissimilarity value. Only required to speed computations if this value is known in advance.
se	standard error estimate of the sample dissimilarity. Only required to speed computations if this value is known in advance.
conf.level	confidence level of the one-sided confidence interval, a numeric value between 0 and 1.
z.conf.level	standard normal (or bootstrap, see arguments below) distribution quantile at the 1 - conf.level value. Only required to speed computations if this value is known in advance. Then, the argument conf.level is ignored.
boot	boolean. If TRUE, z.conf.level is computed by means of a bootstrap approach instead of the asymptotic normal approach. Defaults to FALSE.
nboot	numeric, number of initially planned bootstrap replicates. Ignored if boot == FALSE. Defaults to 10000.
check.table	Boolean. If TRUE (default), argument x is checked to adequately represent a 2x2 contingency table. This checking is performed by means of function nice2x2Table.
y	an object of class "character" representing a vector of gene identifiers (e.g., ENTREZ).

Details

This function computes the upper limit of a one-sided confidence interval for the Sorensen-Dice dissimilarity, given a 2x2 arrangement of frequencies (either implemented as a "table", a "matrix" or a "numeric" object):

$$\begin{matrix} n_{11} & n_{10} \\ n_{01} & n_{00}, \end{matrix}$$

The subindex '11' corresponds to those GO terms enriched in both lists, '01' to terms enriched in the second list but not in the first one, '10' to terms enriched in the first list but not enriched in the second one and '00' corresponds to those GO terms non enriched in both gene lists, i.e., to the double negatives, a value which is ignored in the computations, except if `boot == TRUE`.

In the "numeric" interface, if `length(x) >= 4`, the values are interpreted as $(n_{11}, n_{01}, n_{10}, n_{00})$, always in this order and discarding extra values if necessary.

Arguments `dis`, `se` and `z.conf.level` are not required. If known in advance (e.g., as a consequence of previous computations with the same data), providing its value may speed the computations.

By default, `z.conf.level` corresponds to the 1 - conf.level quantile of a standard normal N(0,1) distribution, as the studentized statistic (^d - d) / ^se) is asymptotically N(0,1). In the studentized statistic, d stands for the "true" Sorensen-Dice dissimilarity, ^d to its sample estimate and ^se for the estimate of its standard error. In fact, the normal is its limiting distribution but, for finite samples, the true sampling distribution may present departures from normality (mainly with some inflation in the left tail). The bootstrap method provides a better approximation to the true sampling distribution. In the bootstrap approach, `nboot` new bootstrap contingency tables are generated from a multinomial distribution with parameters `size` = $n = n_{11} + n_{01} + n_{10} + n_{00}$ and probabilities $(n_{11}/n, n_{01}/n, n_{10}, n_{00}/n)$. Sometimes, some of these generated tables may present so low frequencies of enrichment that make them unable for Sorensen-Dice computations. As a consequence, the number of effective bootstrap samples may be lower than the number of initially planned bootstrap samples `nboot`. Computing in advance the value of argument `z.conf.level` may be a way to cope with these departures from normality, by means of a more adequate quantile function. Alternatively, if `boot == TRUE`, a bootstrap quantile is internally computed.

If `x` is an object of class "character", then `x` (and `y`) must represent two "character" vectors of valid gene identifiers (e.g., ENTREZ). Then the confidence interval for the dissimilarity between lists `x` and `y` is computed, after internally summarizing them as a 2x2 contingency table of joint enrichment. This last operation is performed by function `buildEnrichTable` and "valid gene identifiers (e.g., ENTREZ)" stands for the coherency of these gene identifiers with the arguments `geneUniverse` and `orgPackg` of `buildEnrichTable`, passed by the ellipsis argument `...` in `dUppSorensen`.

In the "list" interface, the argument must be a list of "character" vectors, each one representing a gene list (character identifiers). Then, all pairwise upper limits of the dissimilarity between these gene lists are computed.

In the "tableList" interface, the upper limits are computed over each one of these tables. Given gene lists (i.e. "character" vectors of gene identifiers) l1, l2, ..., lk, an object of class "tableList" (typically constructed by a call to function `buildEnrichTable`) is a list of lists of contingency tables t(i,j) generated from each pair of gene lists i and j, with the following structure:

$l2

$l2$l1$t(2,1)

$l3

$l3$l1$t(3,1), $l3$l2$t(3,2)

...

$lk

lkl1$t(k,1), lkl2$t(k,2), ..., lkl(k-1)t(k,k-1)

Value

In the "table", "matrix", "numeric" and "character" interfaces, the value of the Upper limit of the confidence interval for the Sorensen-Dice dissimilarity. When `boot == TRUE`, this result also haves a an extra attribute: "eff.nboot" which corresponds to the number of effective bootstrap replicats, see the details section. In the "list" and "tableList" interfaces, the result is the symmetric matrix of all pairwise upper limits.

Methods (by class)

- `duppSorensen(table)`: S3 method for class "table"
- `duppSorensen(matrix)`: S3 method for class "matrix"
- `duppSorensen(numeric)`: S3 method for class "numeric"
- `duppSorensen(character)`: S3 method for class "character"
- `duppSorensen(list)`: S3 method for class "list"
- `duppSorensen(tableList)`: S3 method for class "tableList"

See Also

`buildEnrichTable` for constructing contingency tables of mutual enrichment, `nice2x2Table` for checking contingency tables validity, `dSorensen` for computing the Sorensen-Dice dissimilarity, `seSorensen` for computing the standard error of the dissimilarity, `equivTestSorensen` for an equivalence test.

Examples

```
# Gene lists 'atlas' and 'sanger' in 'Cangenes' dataset. Table of joint enrichment
# of GO terms in ontology BP at level 3.
data(cont_atlas.sanger_BP4)
?cont_atlas.sanger_BP4
duppSorensen(cont_atlas.sanger_BP4)
dSorensen(cont_atlas.sanger_BP4) + qnorm(0.95) * seSorensen(cont_atlas.sanger_BP4)
# Using the bootstrap approximation instead of the normal approximation to
# the sampling distribution of (^d - d) / se(^d):
duppSorensen(cont_atlas.sanger_BP4, boot = TRUE)

# Contingency table as a numeric vector:
duppSorensen(c(56, 1, 30, 47))
duppSorensen(c(56, 1, 30))

# Upper confidence limit for the Sorensen-Dice dissimilarity, from scratch,
# directly from two gene lists:
```

```
# (These examples may be considerably time consuming due to many enrichment
# tests to build the contingency tables of mutual enrichment)
# data(allOncoGeneLists)
# ?allOncoGeneLists

# Obtaining ENTREZ identifiers for the gene universe of humans:
# library(org.Hs.eg.db)
# humanEntrezIDs <- keys(org.Hs.eg.db, keytype = "ENTREZID")

# Computing the Upper confidence limit:
# duppSorensen(allOncoGeneLists$atlas, allOncoGeneLists$sanger,
#              onto = "CC", GOLevel = 5,
#              geneUniverse = humanEntrezIDs, orgPackg = "org.Hs.eg.db")
# Even more time consuming (all pairwise values):
# duppSorensen(allOncoGeneLists,
#              onto = "CC", GOLevel = 5,
#              geneUniverse = humanEntrezIDs, orgPackg = "org.Hs.eg.db")
```

`enrichedIn` *This function builds a cross-tabulation of enriched (TRUE) and non-enriched (FALSE) GO terms vs. gene lists*

Description

This function builds a cross-tabulation of enriched (TRUE) and non-enriched (FALSE) GO terms vs. gene lists

Usage

```
enrichedIn(x, ...)

## Default S3 method:
enrichedIn(
  x,
  geneUniverse,
  orgPackg,
  onto,
  GOLevel,
  pAdjustMeth = "BH",
  pvalCutoff = 0.01,
  qvalCutoff = 0.05,
  parallel = FALSE,
  nOfCores = 1,
  onlyEnriched = TRUE,
  keyType = "ENTREZID",
  ...
)
```

```
## S3 method for class 'character'
enrichedIn(
  x,
  geneUniverse,
  orgPackg,
  onto,
  GOLevel,
  pAdjustMeth = "BH",
  pvalCutoff = 0.01,
  qvalCutoff = 0.05,
  parallel = FALSE,
  nOfCores = 1,
  onlyEnriched = TRUE,
  keyType = "ENTREZID",
  ...
)

## S3 method for class 'list'
enrichedIn(
  x,
  geneUniverse,
  orgPackg,
  onto,
  GOLevel,
  pAdjustMeth = "BH",
  pvalCutoff = 0.01,
  qvalCutoff = 0.05,
  parallel = FALSE,
  nOfCores = min(detectCores() - 1, length(x)),
  onlyEnriched = TRUE,
  keyType = "ENTREZID",
  ...
)
```

Arguments

x	either an object of class "character" (or coerzable to "character") or "list". In the "character" interface, these values should represent Entrez gene (or, in general, feature) identifiers. In the "list" interface, each element of the list must be a "character" vector of Entrez identifiers
...	Additional parameters
geneUniverse	character vector containing the universe of genes from where gene lists have been extracted. This vector must be obtained from the annotation package declared in orgPackg. For more details, refer to vignette goSorensen_Introduction.
orgPackg	A string with the name of the genomic annotation package corresponding to a specific species to be analyzed, which must be previously installed and activated. For more details, refer to vignette goSorensen_Introduction.
onto	string describing the ontology. Belongs to c('BP', 'MF', 'CC')

GOLevel	GO level, an integer
pAdjustMeth	string describing the adjust method. Belongs to c('BH', 'BY', 'Bonf')
pvalCutoff	adjusted pvalue cutoff on enrichment tests to report
qvalCutoff	qvalue cutoff on enrichment tests to report as significant. Tests must pass i) pvalueCutoff on unadjusted pvalues, ii) pvalueCutoff on adjusted pvalues and iii) qvalueCutoff on qvalues to be reported
parallel	Logical. Only in "list" interface. Defaults to FALSE, put it at TRUE for parallel computation
nOfCores	Number of cores for parallel computations. Only in "list" interface
onlyEnriched	logical. If TRUE (the default), the returned result only contains those GO terms enriched in almost one of the gene lists
keyType	character keytype of input gene, e.g., ENTREZID or SYMBOL

Details

When the function argument onlyEnriched is FALSE, commonly the result is a sparse but very large object. This function is primarily designed for internal use of function buildEnrichTable, with argument onlyEnriched always put at its default TRUE value. Then calls to enrichedIn result in much more compact objects, in general.

Argument parallel only applies to interface "list". Its default value ís "FALSE" and you may consider the trade of between the time spent in initializing parallelization and the possible time gain when parallelizing. It is difficult to establish a general guideline, but parallelizing is only worthwhile when analyzing many gene lists, on the order of 30 or more, although it depends a lot on each processor.

Value

In the "character" interface, a length k vector of TRUE/FALSE values corresponding to enrichment or not of the GO terms at level 'GOLev' in ontology 'onto'. If 'onlyEnriched' is FALSE, k corresponds to the total number of these GO terms. If 'onlyEnriched' is TRUE (default) k is the number of enriched GO terms (and then all values in the resulting vector are TRUE). In the "list" interface, a logical matrix of TRUE/FALSE values indicating enrichment or not, with k rows and s columns. s is the number of gene lists (the length of "list" 'x'). If 'onlyEnriched' is FALSE, k corresponds to the total number of GO terms at level 'GOLev' in ontology 'onto'. If 'onlyEnriched' is TRUE (default), the resulting matrix only contains the k rows corresponding to GO terms enriched in almost one of these s gene lists. In both interfaces ("character" or "list"), the result also has an attribute (nTerms) with the total number of GO terms at level 'GOLev' in ontology 'onto'.

Methods (by class)

- enrichedIn(default): S3 default method
- enrichedIn(character): S3 method for class "character"
- enrichedIn(list): S3 method for class "list"

Examples

```
# Obtaining ENTREZ identifiers for the gene universe of humans:
library(org.Hs.eg.db)
humanEntrezIDs <- keys(org.Hs.eg.db, keytype = "ENTREZID")

# Gene lists to be explored for enrichment:
data(allOncoGeneLists)
?allOncoGeneLists

# Computing the cross table:
enrichd <- enrichedIn(allOncoGeneLists[["Vogelstein"]],
                      geneUniverse = humanEntrezIDs, orgPackg = "org.Hs.eg.db",
                      onto = "MF", GOLevel = 6)
enrichd

# Cross table of enriched GO terms (GO ontology MF, level 6) for all gene
# lists in 'allOncoGeneLists':
enrichedAllOncoMF.6 <- enrichedIn(allOncoGeneLists,
                          geneUniverse = humanEntrezIDs, orgPackg = "org.Hs.eg.db",
                          onto = "MF", GOLevel = 6)
enrichedAllOncoMF.6
object.size(enrichedAllOncoMF.6)
# How many GO terms were tested for enrichment at ontology MF and level 6:
attr(enrichedAllOncoMF.6, "nTerms")
```

`enrichedInBP4` *Example of the output produced by the function* `enrichedIn`*. It contains exclusively GO terms enriched in at least one list of* `allOncoGeneLists`*, ontology BP, GO-Level 4.*

Description

A matrix with columns representing the gene lists from `allOncoGeneLists`, and rows with GO terms in the BP ontology at GO-Level 4.

This matrix comprises logit values, with `TRUE` indicating that the associated GO term is enriched in the respective list, and `FALSE` indicating that the GO term is not enriched.

This matrix represents the output of the `enrichedIn` function with the argument `onlyEnriched = TRUE` (default), displaying exclusively the GO terms enriched in at least one list (only rows with at least one `TRUE`).

Usage

```
data(enrichedInBP4)
```

Format

An object of class "matrix" "array"

Details

The attribute nTerms of this matrix represents the total number of terms evaluated in the BP ontology at GO-Level 4. The difference between nTerms and the rows of this matrix indicates the number of non-enriched GO terms across all columns (i.e., rows filled with FALSE).

Please, consider this object as an illustrative example only, which is valid exclusively for the allOncoGeneLists data contained in this package. Note that gene lists, GO terms and Bioconductor may change over time. The current version of these results was generated with Bioconductor version 3.20.

---

eqTest_all_BP4 — *Example of the output produced by the function* equivTestSorensen. *It contains all the possible equivalence tests for the lists from* allOncoGeneLists *at level 4 of ontology BP.*

Description

From the seven lists contained in allOncoGeneLists, this object contains the 7(6)/2 = 21 possible outputs for the equivalence tests to compare all possible pairs of lists, using the normal asymptotic distribution.

Usage

```
data(eqTest_all_BP4)
```

Format

An exclusive object from goSorensen of the class "equivSDhtestList"

Details

The parameters considered to execute these tests are: irrelevance limit d0 = 0.4444 and confidence level conf.level = 0.95.

Consider this object only as an illustrative example, which is valid exclusively for the data allOncoGeneLists contained in this package. Note that gene lists, GO terms, and Bioconductor may change over time. The current version of these results were generated with Bioconductor version 3.20.

`eqTest_atlas.sanger_BP4` *Example of the output produced by the function* `equivTestSorensen`. *It contains the equivalence test for comparing two lists at level 4 of ontology BP.*

Description

The output of an equivalence test to detect biological similarity between the lists atlas and sanger from `allOncoGeneLists`, based on the normal asymptotic distribution.

Usage

```
data(eqTest_atlas.sanger_BP4)
```

Format

An exclusive object from ‘goSorensen‘ of the class "equivSDhtest"

Details

The parameters considered to execute this test are: irrelevance limit `d0 = 0.4444` and confidence level `conf.level = 0.95`.

Consider this object only as an illustrative example, which is valid exclusively for the lists atlas and sanger from the data `allOncoGeneLists` contained in this package. Note that gene lists, GO terms, and Bioconductor may change over time. The current version of these results were generated with Bioconductor version 3.20.

`equivTestSorensen` *Equivalence test based on the Sorensen-Dice dissimilarity*

Description

Equivalence test based on the Sorensen-Dice dissimilarity, computed either by an asymptotic normal approach or by a bootstrap approach.

Usage

```
equivTestSorensen(x, ...)

## S3 method for class 'table'
equivTestSorensen(
  x,
  d0 = 1/(1 + 1.25),
  conf.level = 0.95,
```

```
  boot = FALSE,
  nboot = 10000,
  check.table = TRUE,
  ...
)

## S3 method for class 'matrix'
equivTestSorensen(
  x,
  d0 = 1/(1 + 1.25),
  conf.level = 0.95,
  boot = FALSE,
  nboot = 10000,
  check.table = TRUE,
  ...
)

## S3 method for class 'numeric'
equivTestSorensen(
  x,
  d0 = 1/(1 + 1.25),
  conf.level = 0.95,
  boot = FALSE,
  nboot = 10000,
  check.table = TRUE,
  ...
)

## S3 method for class 'character'
equivTestSorensen(
  x,
  y,
  d0 = 1/(1 + 1.25),
  conf.level = 0.95,
  boot = FALSE,
  nboot = 10000,
  check.table = TRUE,
  ...
)

## S3 method for class 'list'
equivTestSorensen(
  x,
  d0 = 1/(1 + 1.25),
  conf.level = 0.95,
  boot = FALSE,
  nboot = 10000,
  check.table = TRUE,
```

```
  ...
)

## S3 method for class 'tableList'
equivTestSorensen(
  x,
  d0 = 1/(1 + 1.25),
  conf.level = 0.95,
  boot = FALSE,
  nboot = 10000,
  check.table = TRUE,
  ...
)
```

Arguments

x	either an object of class "table", "matrix", "numeric", "character", "list" or "tableList". See the details section for more information.
...	extra parameters for function buildEnrichTable.
d0	equivalence threshold for the Sorensen-Dice dissimilarity, d. The null hypothesis states that d >= d0, i.e., inequivalence between the compared gene lists and the alternative that d < d0, i.e., equivalence or dissimilarity irrelevance (up to a level d0).
conf.level	confidence level of the one-sided confidence interval, a value between 0 and 1.
boot	boolean. If TRUE, the confidence interval and the test p-value are computed by means of a bootstrap approach instead of the asymptotic normal approach. Defaults to FALSE.
nboot	numeric, number of initially planned bootstrap replicates. Ignored if boot == FALSE. Defaults to 10000.
check.table	Boolean. If TRUE (default), argument x is checked to adequately represent a 2x2 contingency table (or an aggregate of them) or gene lists producing a correct table. This checking is performed by means of function nice2x2Table.
y	an object of class "character" representing a list of gene identifiers (e.g., ENTREZ).

Details

This function computes either the normal asymptotic or the bootstrap equivalence test based on the Sorensen-Dice dissimilarity, given a 2x2 arrangement of frequencies (either implemented as a "table", a "matrix" or a "numeric" object):

$$\begin{matrix} n_{11} & n_{10} \\ n_{01} & n_{00}, \end{matrix}$$

The subindex '11' corresponds to those GO terms enriched in both lists, '01' to terms enriched in the second list but not in the first one, '10' to terms enriched in the first list but not enriched in

the second one and '00' corresponds to those GO terms non enriched in both gene lists, i.e., to the double negatives, a value which is ignored in the computations.

In the "numeric" interface, if `length(x) >= 4`, the values are interpreted as $(n_{11}, n_{01}, n_{10}, n_{00})$, always in this order and discarding extra values if necessary.

If `x` is an object of class "character", then `x` (and `y`) must represent two "character" vectors of valid gene identifiers (e.g., ENTREZ). Then the equivalence test is performed between `x` and `y`, after internally summarizing them as a 2x2 contingency table of joint enrichment. This last operation is performed by function `buildEnrichTable` and "valid gene identifiers (e.g., ENTREZ)" stands for the coherency of these gene identifiers with the arguments `geneUniverse` and `orgPackg` of `buildEnrichTable`, passed by the ellipsis argument `...` in `equivTestSorensen`.

If `x` is an object of class "list", each of its elements must be a "character" vector of gene identifiers (e.g., ENTREZ). Then all pairwise equivalence tests are performed between these gene lists.

Class "tableList" corresponds to objects representing all mutual enrichment contingency tables generated in a pairwise fashion: Given gene lists l1, l2, ..., lk, an object of class "tableList" (typically constructed by a call to function `buildEnrichTable`) is a list of lists of contingency tables tij generated from each pair of gene lists i and j, with the following structure:

$l2

$l2$l1$t21

$l3

$l3$l1$t31, $l3$l2$t32

...

lkl1$tk1, lkl2$tk2, ..., lkl(k-1)tk(k-1)

If `x` is an object of class "tableList", the test is performed over each one of these tables.

The test is based on the fact that the studentized statistic (^d - d) / ^se is approximately distributed as a standard normal. ^d stands for the sample Sorensen-Dice dissimilarity, d for its true (unknown) value and ^se for the estimate of its standard error. This result is asymptotically correct, but the true distribution of the studentized statistic is not exactly normal for finite samples, with a heavier left tail than expected under the Gaussian model, which may produce some type I error inflation. The bootstrap method provides a better approximation to this distribution. In the bootstrap approach, `nboot` new bootstrap contingency tables are generated from a multinomial distribution with parameters `size` = $n = (n_{11}+n_{01}+n_{10}+n_{00})$ and probabilities $(n_{11}/n, n_{01}/n, n_{10}, n_{00}/n)$. Sometimes, some of these generated tables may present so low frequencies of enrichment that make them unable for Sorensen-Dice computations. As a consequence, the number of effective bootstrap samples may be lower than the number of initially planned ones, `nboot`, but our simulation studies concluded that this makes the test more conservative, less prone to reject a truly false null hypothesis of inequivalence, but in any case protects from inflating the type I error.

In a bootstrap test result, use `getNboot` to access the number of initially planned bootstrap replicates and `getEffNboot` to access the number of finally effective bootstrap replicates.

Value

For all interfaces (except for the "list" and "tableList" interfaces) the result is a list of class "equivSDhtest" which inherits from "htest", with the following components:

statistic the value of the studentized statistic (dSorensen(x) - d0) / seSorensen(x)

p.value the p-value of the test

conf.int the one-sided confidence interval (0, dUpp]

estimate the Sorensen dissimilarity estimate, dSorensen(x)

null.value the value of d0

stderr the standard error of the Sorensen dissimilarity estimate, seSorensen(x), used as denominator in the studentized statistic

alternative a character string describing the alternative hypothesis

method a character string describing the test

data.name a character string giving the names of the data

enrichTab the 2x2 contingency table of joint enrichment whereby the test was based

For the "list" and "tableList" interfaces, the result is an "equivSDhtestList", a list of objects with all pairwise comparisons, each one being an object of "equivSDhtest" class.

Methods (by class)

- `equivTestSorensen(table)`: S3 method for class "table"
- `equivTestSorensen(matrix)`: S3 method for class "matrix"
- `equivTestSorensen(numeric)`: S3 method for class "numeric"
- `equivTestSorensen(character)`: S3 method for class "character"
- `equivTestSorensen(list)`: S3 method for class "list"
- `equivTestSorensen(tableList)`: S3 method for class "tableList"

See Also

`nice2x2Table` for checking and reformatting data, `dSorensen` for computing the Sorensen-Dice dissimilarity, `seSorensen` for computing the standard error of the dissimilarity, `duppSorensen` for the upper limit of a one-sided confidence interval of the dissimilarity. `getTable`, `getPvalue`, `getUpper`, `getSE`, `getNboot` and `getEffNboot` for accessing specific fields in the result of these testing functions. `update` for updating the result of these testing functions with alternative equivalence limits, confidence levels or to convert a normal result in a bootstrap result or the reverse.

Examples

```
# Gene lists 'atlas' and 'sanger' in 'allOncoGeneLists' dataset. Table of joint enrichment
# of GO terms in ontology BP at level 4.
data(cont_atlas.sanger_BP4)
cont_atlas.sanger_BP4
equivTestSorensen(cont_atlas.sanger_BP4)
# Bootstrap test:
equivTestSorensen(cont_atlas.sanger_BP4, boot = TRUE)

# Equivalence tests from scratch, directly from gene lists:
# (These examples may be considerably time consuming due to many enrichment
# tests to build the contingency tables of mutual enrichment)
# data(allOncoGeneLists)
# ?allOncoGeneLists
```

```
# Obtaining ENTREZ identifiers for the gene universe of humans:
library(org.Hs.eg.db)
humanEntrezIDs <- keys(org.Hs.eg.db, keytype = "ENTREZID")

# Computing the equivalence test:
# equivTestSorensen(allOncoGeneLists$atlas, allOncoGeneLists$sanger,
#                   geneUniverse = humanEntrezIDs, orgPackg = "org.Hs.eg.db",
#                   onto = "BP", GOLevel = 4)
# Bootstrap instead of normal approximation test:
# equivTestSorensen(allOncoGeneLists$atlas, allOncoGeneLists$sanger,
#                   geneUniverse = humanEntrezIDs, orgPackg = "org.Hs.eg.db",
#                   onto = "BP", GOLevel = 4,
#                   boot = TRUE)

# Essentially, the above code makes:
# ccont_atlas.sanger_BP4 <- buildEnrichTable(allOncoGeneLists$atlas, allOncoGeneLists$sanger,
#                               geneUniverse = humanEntrezIDs, orgPackg = "org.Hs.eg.db",
#                                    onto = "BP", GOLevel = 4)
# ccont_atlas.sanger_BP4
# equivTestSorensen(ccont_atlas.sanger_BP4)
# equivTestSorensen(ccont_atlas.sanger_BP4, boot = TRUE)
# (Note that building first the contingency table may be advantageous to save time!)
# The object cont_atlas.sanger_BP4 and ccont_atlas.sanger_BP4 are exactly the same

# All pairwise equivalence tests:
# equivTestSorensen(allOncoGeneLists,
#                   geneUniverse = humanEntrezIDs, orgPackg = "org.Hs.eg.db",
#                   onto = "BP", GOLevel = 4)


# Equivalence test on a contingency table represented as a numeric vector:
equivTestSorensen(c(56, 1, 30, 47))
equivTestSorensen(c(56, 1, 30, 47), boot = TRUE)
equivTestSorensen(c(56, 1, 30))
# Error: all frequencies are needed for bootstrap:
try(equivTestSorensen(c(56, 1, 30), boot = TRUE), TRUE)
```

`fullEnrichedInBP4` *Example of the output produced by the function* `enrichedIn`*. It contains all the GO terms enriched or not-enriched in the lists of* `allOncoGeneLists`*, ontology BP, GO-Level 4.*

Description

A matrix with columns representing the gene lists from `allOncoGeneLists`, and rows with GO terms in the BP ontology at GO-Level 4.

This matrix comprises logit values, with `TRUE` indicating that the associated GO term is enriched in the respective list, and `FALSE` indicating that the GO term is not enriched.

This matrix represents the output of the enrichedIn function with the argument onlyEnriched = FALSE. The rows of this matrix display all the GO terms involved in the BP ontology at GO-Level 4.

Usage

```
data(fullEnrichedInBP4)
```

Format

An object of class "matrix" "array"

Details

The attribute nTerms indicates the total number of GO terms evaluated in the BP ontology, GO-Level 4. For this particular case, nTerms matches with the number of rows of the matrix

Please, consider this object as an illustrative example only, which is valid exclusively for the allOncoGeneLists data contained in this package. Please note that gene lists, GO terms and Bioconductor may change over time. The current version of these results was generated with Bioconductor version 3.20.

---

getDissimilarity *Access to the estimated Sorensen-Dice dissimilarity in one or more equivalence test results*

Description

Given objects representing the result(s) of one or more equivalence tests (classes "equivSDhtest", "equivSDhtestList" or "allEquivSDtest", i.e., the result of functions 'equivTestSorensen' and 'allEquivTestSorensen') this function returns the estimated dissimilarities in the tests.

Usage

```
getDissimilarity(x, ...)

## S3 method for class 'equivSDhtest'
getDissimilarity(x, ...)

## S3 method for class 'equivSDhtestList'
getDissimilarity(x, simplify = TRUE, ...)

## S3 method for class 'AllEquivSDhtest'
getDissimilarity(x, onto, GOLevel, listNames, simplify = TRUE, ...)
```

Arguments

x	an object of class "equivSDhtest" or "equivSDhtestList" or "allEquivSDtest".
...	Additional parameters.
simplify	logical, if TRUE the result is simplified, e.g., returning a vector instead of a matrix.
onto	character, a vector with one or more of "BP", "CC" or "MF", ontologies to access.
GOLevel	numeric or character, a vector with one or more GO levels to access. See the details section and the examples.
listNames	character(2), the names of a pair of gene lists.

Details

Argument GOLevel can be of class "character" or "numeric". In the first case, the GO levels must be specified like "level 6" or c("level 4", "level 5", "level 6") In the second case ("numeric"), the GO levels must be specified like6 or seq.int(4,6).

Value

When x is an object of class "equivSDhtest" (i.e., the result of a single equivalence test), the returned value is a single numeric value, the Sorensen-Dice dissimilarity. For an object of class "equivSDhtestList" (i.e. all pairwise tests for a set of gene lists), if simplify = TRUE (the default), the resulting value is a vector with the dissimilarities in all those tests, or the symmetric matrix of all dissimilarities if simplify = TRUE. If x is an object of class "allEquivSDtest" (i.e., the test iterated along GO ontologies and levels), the preceding result is returned in the form of a list along the ontologies, levels and pairs of gene lists specified by the arguments onto, GOlevel and listNames (or all present in x for missing arguments).

Methods (by class)

- getDissimilarity(equivSDhtest): S3 method for class "equivSDhtest"
- getDissimilarity(equivSDhtestList): S3 method for class "equivSDhtestList"
- getDissimilarity(AllEquivSDhtest): S3 method for class "AllEquivSDhtest"

Examples

```
# Dataset 'eqTest_atlas.sanger_BP4' contains the result of the equivalence test between gene lists
# 'sanger' and 'atlas', at level 4 of the BP ontology:
data(eqTest_atlas.sanger_BP4)
eqTest_atlas.sanger_BP4
class(eqTest_atlas.sanger_BP4)
# This may correspond to the result of code like:
# eqTest_atlas.sanger_BP4 <- equivTestSorensen(
#   allOncoGeneLists[["sanger"]], allOncoGeneLists[["atlas"]],
#   geneUniverse = humanEntrezIDs, orgPackg = "org.Hs.eg.db",
#   onto = "BP", GOLevel = 4, listNames = c("sanger", "atlas"))
# (But results may vary according to GO updating)
getDissimilarity(eqTest_atlas.sanger_BP4)
```

```
# All pairwise equivalence tests at level 4 of the BP ontology:
data(eqTest_all_BP4)
?eqTest_all_BP4
class(eqTest_all_BP4)
# This may correspond to a call like:
# eqTest_all_BP4 <- equivTestSorensen(allOncoGeneLists,
#                           geneUniverse = humanEntrezIDs, orgPackg = "org.Hs.eg.db",
#                           onto = "BP", GOLevel = 4)
getDissimilarity(eqTest_all_BP4)
getDissimilarity(eqTest_all_BP4, simplify = FALSE)

# Equivalence test iterated over all GO ontologies and levels 3 to 10:
data(allEqTests)
?allEqTests
class(allEqTests)
# This may correspond to code like:
# (By default, the tests are iterated over all GO ontologies and for levels 3 to 10)
# allEqTests <- allEquivTestSorensen(allOncoGeneLists,
#                                          geneUniverse = humanEntrezIDs,
#                                          orgPackg = "org.Hs.eg.db")
# All Sorensen-Dice dissimilarities:
getDissimilarity(allEqTests)
getDissimilarity(allEqTests, simplify = FALSE)

# Dissimilarities only for some GO ontologies, levels or pairs of gene lists:
getDissimilarity(allEqTests, GOLevel = "level 6")
getDissimilarity(allEqTests, GOLevel = 6)
getDissimilarity(allEqTests, GOLevel = seq.int(4,6))
getDissimilarity(allEqTests, GOLevel = "level 6", simplify = FALSE)
getDissimilarity(allEqTests, GOLevel = "level 6", listNames = c("waldman", "sanger"))
getDissimilarity(allEqTests, GOLevel = seq.int(4,6), onto = "BP")
getDissimilarity(allEqTests, GOLevel = seq.int(4,6), onto = "BP", simplify = FALSE)
getDissimilarity(allEqTests, GOLevel = "level 6", onto = "BP",
                 listNames = c("atlas", "sanger"))
getDissimilarity(allEqTests$BP$`level 4`)
```

`getEffNboot` *Access to the number of effective bootstrap replicates in one or more equivalence test results (only for their bootstrap version)*

Description

Given objects representing the result(s) of one or more equivalence tests (classes "equivSDhtest", "equivSDhtestList" or "allEquivSDtest", i.e., the result of functions 'equivTestSorensen' and 'allEquivTestSorensen'), this function returns the number of effective bootstrap replicates. Obviously, this only applies to calls of these functions with the parameter boot = TRUE, otherwise it returns a NA value. See the details section for further explanation.

Usage

```
getEffNboot(x, ...)

## S3 method for class 'equivSDhtest'
getEffNboot(x, ...)

## S3 method for class 'equivSDhtestList'
getEffNboot(x, simplify = TRUE, ...)

## S3 method for class 'AllEquivSDhtest'
getEffNboot(x, onto, GOLevel, listNames, simplify = TRUE, ...)
```

Arguments

x	an object of class "equivSDhtest" or "equivSDhtestList" or "allEquivSDtest".
...	Additional parameters.
simplify	logical, if TRUE the result is simplified, e.g., returning a vector instead of a matrix.
onto	character, a vector with one or more of "BP", "CC" or "MF", ontologies to access.
GOLevel	numeric or character, a vector with one or more GO levels to access. See the details section and the examples.
listNames	character(2), the names of a pair of gene lists.

Details

In the bootstrap version of the equivalence test, resampling is performed generating new bootstrap contingency tables from a multinomial distribution based on the "real", observed, frequencies of mutual enrichment. In some bootstrap resamples, the generated contingency table of mutual enrichment may have very low frequencies of enrichment, which makes it unable for Sorensen-Dice computations. Then, the number of effective bootstrap resamples may be lower than those initially planned. To get the number of initially planned bootstrap resamples use function `getNboot`.

Argument `GOLevel` can be of class "character" or "numeric". In the first case, the GO levels must be specified like `"level 6"` or `c("level 4", "level 5", "level 6")` In the second case ("numeric"), the GO levels must be specified like`6` or `seq.int(4,6)`.

Value

When `x` is an object of class "equivSDhtest" (i.e., the result of a single equivalence test), the returned value is a single numeric value, the number of effective bootstrap replicates, or `NA` if bootstrapping has not been performed. For an object of class "equivSDhtestList" (i.e. all pairwise tests for a set of gene lists), if `simplify = TRUE` (the default), the resulting value is a vector with the number of effective bootstrap replicates in all those tests, or the symmetric matrix of all these values if `simplify = TRUE`. If `x` is an object of class "allEquivSDtest" (i.e., the test iterated along GO ontologies and levels), the preceding result is returned in the form of a list along the ontologies, levels and pairs of gene lists specified by the arguments `onto`, `GOlevel` and `listNames` (or all present in `x` for missing arguments).

Methods (by class)

- `getEffNboot(equivSDhtest)`: S3 method for class "equivSDhtest"
- `getEffNboot(equivSDhtestList)`: S3 method for class "equivSDhtestList"
- `getEffNboot(AllEquivSDhtest)`: S3 method for class "AllEquivSDhtest"

See Also

`getNboot`

Examples

```
# Dataset 'allOncoGeneLists' contains the result of the equivalence test between gene lists
# 'sanger' and 'atlas', at level 4 of the BP ontology:
data(eqTest_atlas.sanger_BP4)
eqTest_atlas.sanger_BP4
class(eqTest_atlas.sanger_BP4)
# This may correspond to the result of code like:
# eqTest_atlas.sanger_BP4 <- equivTestSorensen(
#   allOncoGeneLists[["sanger"]], allOncoGeneLists[["atlas"]],
#   geneUniverse = humanEntrezIDs, orgPackg = "org.Hs.eg.db",
#   onto = "BP", GOLevel = 4, listNames = c("sanger", "atlas"))
#
# (But results may vary according to GO updating)

# Not a bootstrap test, first upgrade to a bootstrap test:
boot.sanger_atlas.BP.4 <- upgrade(eqTest_atlas.sanger_BP4, boot = TRUE)

#getEffNboot(eqTest_atlas.sanger_BP4)
getEffNboot(boot.sanger_atlas.BP.4)
getNboot(boot.sanger_atlas.BP.4)

# All pairwise equivalence tests at level 4 of the BP ontology
data(eqTest_all_BP4)
?eqTest_all_BP4
class(eqTest_all_BP4)
# This may correspond to a call like:
# eqTest_all_BP4 <- equivTestSorensen(allOncoGeneLists,
#                           geneUniverse = humanEntrezIDs, orgPackg = "org.Hs.eg.db",
#                           onto = "BP", GOLevel = 4)
boot.BP.4 <- upgrade(eqTest_all_BP4, boot = TRUE)
getEffNboot(eqTest_all_BP4)
getEffNboot(boot.BP.4)
getNboot(boot.BP.4)
getEffNboot(boot.BP.4, simplify = FALSE)

# Bootstrap equivalence test iterated over all GO ontologies and levels 3 to 10.
# data(allEqTests)
# ?allEqTests
# class(allEqTests)
# This may correspond to code like:
# (By default, the tests are iterated over all GO ontologies and for levels 3 to 10)
```

```
# allEqTests <- allEquivTestSorensen(allOncoGeneLists,
#                                            geneUniverse = humanEntrezIDs,
#                                            orgPackg = "org.Hs.eg.db",
#                                            boot = TRUE)
# boot.allEqTests <- upgrade(allEqTests, boot = TRUE)
# Number of effective bootstrap replicates for all tests:
# getEffNboot(boot.allEqTests)
# getEffNboot(boot.allEqTests, simplify = FALSE)

# Number of effective bootstrap replicates for specific GO ontologies, levels or pairs
# of gene lists:
# getEffNboot(boot.allEqTests, GOLevel = "level 6")
# getEffNboot(boot.allEqTests, GOLevel = 6)
# getEffNboot(boot.allEqTests, GOLevel = seq.int(4,6))
# getEffNboot(boot.allEqTests, GOLevel = "level 6", simplify = FALSE)
# getEffNboot(boot.allEqTests, GOLevel = "level 6", listNames = c("waldman", "sanger"))
# getEffNboot(boot.allEqTests, GOLevel = seq.int(4,6), onto = "BP")
# getEffNboot(boot.allEqTests, GOLevel = seq.int(4,6), onto = "BP", simplify = FALSE)
# getEffNboot(boot.allEqTests, GOLevel = "level 6", onto = "BP",
#             listNames = c("atlas", "sanger"))
# getEffNboot(boot.allEqTests$BP$`level 4`)
```

---

getNboot *Access to the number of initially planned bootstrap replicates in one or more equivalence test results (only in their bootstrap version)*

Description

Given objects representing the result(s) of one or more equivalence tests (classes "equivSDhtest", "equivSDhtestList" or "allEquivSDtest", i.e., the result of functions 'equivTestSorensen' and 'allEquivTestSorensen' with the parameter boot = TRUE), this function returns the number of initially planned bootstrap replicates in these equivalence tests, which may be greater than the number of finally effective or valid bootstrap replicates. See the details section for more information on this.

Usage

```
getNboot(x, ...)

## S3 method for class 'equivSDhtest'
getNboot(x, ...)

## S3 method for class 'equivSDhtestList'
getNboot(x, simplify = TRUE, ...)

## S3 method for class 'AllEquivSDhtest'
getNboot(x, onto, GOLevel, listNames, simplify = TRUE, ...)
```

Arguments

x	an object of class "equivSDhtest" or "equivSDhtestList" or "allEquivSDtest".
...	Additional parameters.
simplify	logical, if TRUE the result is simplified, e.g., returning a vector instead of a matrix.
onto	character, a vector with one or more of "BP", "CC" or "MF", ontologies to access.
GOLevel	numeric or character, a vector with one or more GO levels to access. See the details section and the examples.
listNames	character(2), the names of a pair of gene lists.

Details

In the bootstrap version of the equivalence test, resampling is performed generating new bootstrap contingency tables from a multinomial distribution based on the "real", observed, frequencies of mutual enrichment. In some bootstrap iterations, the generated contingency table of mutual enrichment may have very low frequencies of enrichment, which makes it unable for Sorensen-Dice computations. Then, the number of effective bootstrap resamples may be lower than those initially planned. To get the number of effective bootstrap resamples use function `getEffNboot`.

Argument `GOLevel` can be of class "character" or "numeric". In the first case, the GO levels must be specified like `"level 6"` or `c("level 4", "level 5", "level 6")` In the second case ("numeric"), the GO levels must be specified like`6` or `seq.int(4,6)`.

Value

When `x` is an object of class "equivSDhtest" (i.e., the result of a single equivalence test), the returned value is a single numeric value, the number of initially planned bootstrap replicates, or `NA` if bootstrapping has not been performed. For an object of class "equivSDhtestList" (i.e. all pairwise tests for a set of gene lists), if `simplify = TRUE` (the default), the resulting value is a vector with the number of initially bootstrap replicates in all those tests, or the symmetric matrix of all these values if `simplify = TRUE`. If `x` is an object of class "allEquivSDtest" (i.e., the test iterated along GO ontologies and levels), the preceding result is returned in the form of a list along the ontologies, levels and pairs of gene lists specified by the arguments `onto`, `GOlevel` and `listNames` (or all present in `x` for missing arguments).

Methods (by class)

- `getNboot(equivSDhtest)`: S3 method for class "equivSDhtest"
- `getNboot(equivSDhtestList)`: S3 method for class "equivSDhtestList"
- `getNboot(AllEquivSDhtest)`: S3 method for class "AllEquivSDhtest"

See Also

`getEffNboot`

Examples

```
# Dataset 'allOncoGeneLists' contains the result of the equivalence test between gene lists
# 'sanger' and 'atlas', at level 4 of the BP ontology:
data(eqTest_atlas.sanger_BP4)
eqTest_atlas.sanger_BP4
class(eqTest_atlas.sanger_BP4)
# This may correspond to the result of code like:
# eqTest_atlas.sanger_BP4 <- equivTestSorensen(
#   allOncoGeneLists[["sanger"]], allOncoGeneLists[["atlas"]],
#   geneUniverse = humanEntrezIDs, orgPackg = "org.Hs.eg.db",
#   onto = "BP", GOLevel = 4, listNames = c("sanger", "atlas"))
#
# (But results may vary according to GO updating)

# Not a bootstrap test, first upgrade to a bootstrap test:
boot.eqTest_atlas.sanger_BP4 <- upgrade(eqTest_atlas.sanger_BP4, boot = TRUE)

getNboot(eqTest_atlas.sanger_BP4)
getNboot(boot.eqTest_atlas.sanger_BP4)

# All pairwise equivalence tests at level 4 of the BP ontology
data(eqTest_all_BP4)
?eqTest_all_BP4
class(eqTest_all_BP4)
# This may correspond to a call like:
# eqTest_all_BP4 <- equivTestSorensen(allOncoGeneLists,
#                            geneUniverse = humanEntrezIDs, orgPackg = "org.Hs.eg.db",
#                            onto = "BP", GOLevel = 4)
boot.eqTest_all_BP4 <- upgrade(eqTest_all_BP4, boot = TRUE)
getNboot(eqTest_all_BP4)
getNboot(boot.eqTest_all_BP4)
getNboot(boot.eqTest_all_BP4, simplify = FALSE)

# Bootstrap equivalence test iterated over all GO ontologies and levels 3 to 10.
# data(allEqTests)
# ?allEqTests
# class(allEqTests)
# This may correspond to code like:
# (By default, the tests are iterated over all GO ontologies and for levels 3 to 10)
# allEqTests <- allEquivTestSorensen(allOncoGeneLists,
#                                            geneUniverse = humanEntrezIDs,
#                                            orgPackg = "org.Hs.eg.db",
#                                            boot = TRUE)
# boot.allEqTests <- upgrade(allEqTests, boot = TRUE)
# All numbers of bootstrap replicates:
# getNboot(boot.allEqTests)
# getNboot(boot.allEqTests, simplify = FALSE)

# Number of bootstrap replicates for specific GO ontologies, levels or pairs of gene lists:
# getNboot(boot.allEqTests, GOLevel = "level 6")
# getNboot(boot.allEqTests, GOLevel = 6)
# getNboot(boot.allEqTests, GOLevel = seq.int(4,6))
```

```
# getNboot(boot.allEqTests, GOLevel = "level 6", simplify = FALSE)
# getNboot(boot.allEqTests, GOLevel = "level 6", listNames = c("atlas", "sanger"))
# getNboot(boot.allEqTests, GOLevel = seq.int(4,6), onto = "BP")
# getNboot(boot.allEqTests, GOLevel = seq.int(4,6), onto = "BP", simplify = FALSE)
# getNboot(boot.allEqTests, GOLevel = "level 6", onto = "BP",
#          listNames = c("waldman", "sanger"))
# getNboot(boot.allEqTests$BP$`level 4`)
```

`getPvalue` *Access to the p-value of one or more equivalence test results*

Description

Given objects representing the result(s) of one or more equivalence tests (classes "equivSDhtest", "equivSDhtestList" or "allEquivSDtest", i.e., the result of functions 'equivTestSorensen' and 'allEquivTestSorensen') this function returns the p-values of the tests.

Usage

```
getPvalue(x, ...)

## S3 method for class 'equivSDhtest'
getPvalue(x, ...)

## S3 method for class 'equivSDhtestList'
getPvalue(x, simplify = TRUE, ...)

## S3 method for class 'AllEquivSDhtest'
getPvalue(x, onto, GOLevel, listNames, simplify = TRUE, ...)
```

Arguments

`x`	an object of class "equivSDhtest" or "equivSDhtestList" or "allEquivSDtest".
`...`	Additional parameters.
`simplify`	logical, if TRUE the result is simplified, e.g., returning a vector instead of a matrix.
`onto`	character, a vector with one or more of "BP", "CC" or "MF", ontologies to access.
`GOLevel`	numeric or character, a vector with one or more GO levels to access. See the details section and the examples.
`listNames`	character(2), the names of a pair of gene lists.

Details

Argument `GOLevel` can be of class "character" or "numeric". In the first case, the GO levels must be specified like `"level 6"` or `c("level 4", "level 5", "level 6")` In the second case ("numeric"), the GO levels must be specified like`6` or `seq.int(4,6)`.

Value

When x is an object of class "equivSDhtest" (i.e., the result of a single equivalence test), the returned value is a single numeric value, the test p-value. For an object of class "equivSDhtestList" (i.e. all pairwise tests for a set of gene lists), if `simplify = TRUE` (the default), the resulting value is a vector with the p-values in all those tests, or the symmetric matrix of all p-values if `simplify = TRUE`. If x is an object of class "allEquivSDtest" (i.e., the test iterated along GO ontologies and levels), the preceding result is returned in the form of a list along the ontologies, levels and pairs of gene lists specified by the arguments `onto`, `GOlevel` and `listNames` (or all present in x for missing arguments).

Methods (by class)

- `getPvalue(equivSDhtest)`: S3 method for class "equivSDhtest"
- `getPvalue(equivSDhtestList)`: S3 method for class "equivSDhtestList"
- `getPvalue(AllEquivSDhtest)`: S3 method for class "AllEquivSDhtest"

Examples

```
# Dataset 'allOncoGeneLists' contains the result of the equivalence test between gene lists
# 'sanger' and 'atlas', at level 4 of the BP ontology:
data(eqTest_atlas.sanger_BP4)
eqTest_atlas.sanger_BP4
class(eqTest_atlas.sanger_BP4)
# This may correspond to the result of code like:
# eqTest_atlas.sanger_BP4 <- equivTestSorensen(
#   allOncoGeneLists[["sanger"]], allOncoGeneLists[["atlas"]],
#   geneUniverse = humanEntrezIDs, orgPackg = "org.Hs.eg.db",
#   onto = "BP", GOLevel = 4, listNames = c("sanger", "atlas"))
# (But results may vary according to GO updating)
getPvalue(eqTest_atlas.sanger_BP4)

# All pairwise equivalence tests at level 4 of the BP ontology
data(eqTest_all_BP4)
?eqTest_all_BP4
class(eqTest_all_BP4)
# This may correspond to a call like:
# eqTest_all_BP4 <- equivTestSorensen(allOncoGeneLists,
#                            geneUniverse = humanEntrezIDs, orgPackg = "org.Hs.eg.db",
#                            onto = "BP", GOLevel = 4)
getPvalue(eqTest_all_BP4)
getPvalue(eqTest_all_BP4, simplify = FALSE)

# Equivalence test iterated over all GO ontologies and levels 3 to 10:
data(allEqTests)
?allEqTests
class(allEqTests)
# This may correspond to code like:
# (By default, the tests are iterated over all GO ontologies and for levels 3 to 10)
# allEqTests <- allEquivTestSorensen(allOncoGeneLists,
#                                             geneUniverse = humanEntrezIDs,
#                                             orgPackg = "org.Hs.eg.db")
```

```
# All p-values:
getPvalue(allEqTests)
getPvalue(allEqTests, simplify = FALSE)

# P-values only for some GO ontologies, levels or pairs of gene lists:
getPvalue(allEqTests, GOLevel = "level 6")
getPvalue(allEqTests, GOLevel = 6)
getPvalue(allEqTests, GOLevel = seq.int(4,6))
getPvalue(allEqTests, GOLevel = "level 6", simplify = FALSE)
getPvalue(allEqTests, GOLevel = "level 6", listNames = c("atlas", "sanger"))
getPvalue(allEqTests, GOLevel = seq.int(4,6), onto = "BP")
getPvalue(allEqTests, GOLevel = seq.int(4,6), onto = "BP", simplify = FALSE)
getPvalue(allEqTests, GOLevel = "level 6", onto = "BP",
          listNames = c("waldman", "sanger"))
getPvalue(allEqTests$BP$`level 4`)
```

`getSE` *Access to the estimated standard error of the sample Sorensen-Dice dissimilarity in one or more equivalence test results*

Description

Given objects representing the result(s) of one or more equivalence tests (classes "equivSDhtest", "equivSDhtestList" or "allEquivSDtest", i.e., the result of functions 'equivTestSorensen' and 'allEquivTestSorensen') this function returns the estimated standard errors of the sample dissimilarities in the tests.

Usage

```
getSE(x, ...)

## S3 method for class 'equivSDhtest'
getSE(x, ...)

## S3 method for class 'equivSDhtestList'
getSE(x, simplify = TRUE, ...)

## S3 method for class 'AllEquivSDhtest'
getSE(x, onto, GOLevel, listNames, simplify = TRUE, ...)
```

Arguments

`x`	an object of class "equivSDhtest" or "equivSDhtestList" or "allEquivSDtest".
`...`	additional parameters.
`simplify`	logical, if TRUE the result is simplified, e.g., returning a vector instead of a matrix.

onto	character, a vector with one or more of "BP", "CC" or "MF", ontologies to access.
GOLevel	numeric or character, a vector with one or more GO levels to access. See the details section and the examples.
listNames	character(2), the names of a pair of gene lists.

Details

Argument GOLevel can be of class "character" or "numeric". In the first case, the GO levels must be specified like "level 6" or c("level 4", "level 5", "level 6") In the second case ("numeric"), the GO levels must be specified like6 or seq.int(4,6).

Value

When x is an object of class "equivSDhtest" (i.e., the result of a single equivalence test), the returned value is a single numeric value, the standard error of the Sorensen-Dice dissimilarity estimate. For an object of class "equivSDhtestList" (i.e. all pairwise tests for a set of gene lists), if simplify = TRUE (the default), the resulting value is a vector with the dissimilarity standard errors in all those tests, or the symmetric matrix of all these values if simplify = TRUE. If x is an object of class "allEquivSDtest" (i.e., the test iterated along GO ontologies and levels), the preceding result is returned in the form of a list along the ontologies, levels and pairs of gene lists specified by the arguments onto, GOlevel and listNames (or all present in x for missing arguments).

Methods (by class)

- getSE(equivSDhtest): S3 method for class "equivSDhtest"
- getSE(equivSDhtestList): S3 method for class "equivSDhtestList"
- getSE(AllEquivSDhtest): S3 method for class "AllEquivSDhtest"

Examples

```
# Dataset 'allOncoGeneLists' contains the result of the equivalence test between gene lists
# 'sanger' and 'atlas', at level 4 of the BP ontology:
data(eqTest_atlas.sanger_BP4)
eqTest_atlas.sanger_BP4
class(eqTest_atlas.sanger_BP4)
# This may correspond to the result of code like:
# eqTest_atlas.sanger_BP4 <- equivTestSorensen(
#   allOncoGeneLists[["sanger"]], allOncoGeneLists[["atlas"]],
#   geneUniverse = humanEntrezIDs, orgPackg = "org.Hs.eg.db",
#   onto = "BP", GOLevel = 4, listNames = c("sanger", "atlas"))
# (But results may vary according to GO updating)
getSE(eqTest_atlas.sanger_BP4)

# All pairwise equivalence tests at level 4 of the BP ontology:
data(eqTest_all_BP4)
?eqTest_all_BP4
class(eqTest_all_BP4)
# This may correspond to a call like:
# eqTest_all_BP4 <- equivTestSorensen(allOncoGeneLists,
```

```
#                           geneUniverse = humanEntrezIDs, orgPackg = "org.Hs.eg.db",
#                           onto = "BP", GOLevel = 4)
getSE(eqTest_all_BP4)
getSE(eqTest_all_BP4, simplify = FALSE)

# Equivalence test iterated over all GO ontologies and levels 3 to 10:
data(allEqTests)
?allEqTests
class(allEqTests)
# This may correspond to code like:
# (By default, the tests are iterated over all GO ontologies and for levels 3 to 10)
# allEqTests <- allEquivTestSorensen(allOncoGeneLists,
#                                          geneUniverse = humanEntrezIDs,
#                                          orgPackg = "org.Hs.eg.db")
# All standard errors of the Sorensen-Dice dissimilarity estimates:
getSE(allEqTests)
getSE(allEqTests, simplify = FALSE)

# Standard errors for some GO ontologies, levels or pairs of gene lists:
getSE(allEqTests, GOLevel = "level 6")
getSE(allEqTests, GOLevel = 6)
getSE(allEqTests, GOLevel = seq.int(4,6))
getSE(allEqTests, GOLevel = "level 6", simplify = FALSE)
getSE(allEqTests, GOLevel = "level 6", listNames = c("atlas", "sanger"))
getSE(allEqTests, GOLevel = seq.int(4,6), onto = "BP")
getSE(allEqTests, GOLevel = seq.int(4,6), onto = "BP", simplify = FALSE)
getSE(allEqTests, GOLevel = "level 6", onto = "BP",
      listNames = c("waldman", "sanger"))
getSE(allEqTests$BP$`level 4`)
```

`getTable` *Access to the contingency table of mutual enrichment of one or more equivalence test results*

Description

Given objects representing the result(s) of one or more equivalence tests (classes "equivSDhtest", "equivSDhtestList" or "allEquivSDtest", i.e., the result of functions 'equivTestSorensen' and 'allEquivTestSorensen') this function returns the contingency tables from which the tests were performed.

Usage

```
getTable(x, ...)

## S3 method for class 'equivSDhtest'
getTable(x, ...)
```

```
## S3 method for class 'equivSDhtestList'
getTable(x, ...)

## S3 method for class 'AllEquivSDhtest'
getTable(x, onto, GOLevel, listNames, ...)
```

Arguments

x	an object of class "equivSDhtest" or "equivSDhtestList" or "allEquivSDtest".
...	Additional parameters.
onto	character, a vector with one or more of "BP", "CC" or "MF", ontologies to access.
GOLevel	numeric or character, a vector with one or more GO levels to access. See the details section and the examples.
listNames	character(2), the names of a pair of gene lists.

Details

Argument GOLevel can be of class "character" or "numeric". In the first case, the GO levels must be specified like "level 6" or c("level 4", "level 5", "level 6") In the second case ("numeric"), the GO levels must be specified like6 or 4:6.

Value

An object of class "table", the 2x2 enrichment contingeny table of mutual enrichment in two gene lists, built to perform the equivalence test based on the Sorensen-Dice dissimilarity.

When x is an object of class "equivSDhtest" (i.e., the result of a single equivalence test), the returned value is an object of class "table", the 2x2 enrichment contingeny table of mutual enrichment in two gene lists, built to perform the equivalence test based on the Sorensen-Dice dissimilarity. For an object of class "equivSDhtestList" (i.e. all pairwise tests for a set of gene lists), the resulting value is a list with all the tables built in all those tests. If x is an object of class "allEquivSDtest" (i.e., the test iterated along GO ontologies and levels), the preceding result is returned as a list along the ontologies, levels and pairs of gene lists specified by the arguments onto, GOlevel and listNames (or all ontologies, levels or pairs of gene lists present in x if one or more of these arguments are missing).

Methods (by class)

- getTable(equivSDhtest): S3 method for class "equivSDhtest"
- getTable(equivSDhtestList): S3 method for class "equivSDhtestList"
- getTable(AllEquivSDhtest): S3 method for class "AllEquivSDhtest"

Examples

```
# Dataset 'allOncoGeneLists' contains the result of the equivalence test between gene lists
# 'sanger' and 'atlas', at level 4 of the BP ontology:
data(eqTest_atlas.sanger_BP4)
```

```
eqTest_atlas.sanger_BP4
class(eqTest_atlas.sanger_BP4)
# This may correspond to the result of code like:
# eqTest_atlas.sanger_BP4 <- equivTestSorensen(
#   allOncoGeneLists[["sanger"]], allOncoGeneLists[["atlas"]],
#   geneUniverse = humanEntrezIDs, orgPackg = "org.Hs.eg.db",
#   onto = "BP", GOLevel = 4, listNames = c("sanger", "atlas"))
# (But results may vary according to GO updating)
getTable(eqTest_atlas.sanger_BP4)

# All pairwise equivalence tests at level 4 of the BP ontology
data(eqTest_all_BP4)
?eqTest_all_BP4
class(eqTest_all_BP4)
# This may correspond to a call like:
# eqTest_all_BP4 <- equivTestSorensen(allOncoGeneLists,
#                           geneUniverse = humanEntrezIDs, orgPackg = "org.Hs.eg.db",
#                           onto = "BP", GOLevel = 4)
getTable(eqTest_all_BP4)

# Equivalence test iterated over all GO ontologies and levels 3 to 10:
data(allEqTests)
?allEqTests
class(allEqTests)
# This may correspond to code like:
# allEqTests <- allEquivTestSorensen(allOncoGeneLists,
#                                          geneUniverse = humanEntrezIDs,
#                                          orgPackg = "org.Hs.eg.db")
# (By default, the tests are iterated over all GO ontologies and for levels 3 to 10)
# All 2x2 contingecy tables of joint enrichment:
getTable(allEqTests)
# Contingency tables only for some GO ontologies, levels or pairs of gene lists:
getTable(allEqTests, GOLevel = "level 6")
getTable(allEqTests, GOLevel = 6)
getTable(allEqTests, GOLevel = seq.int(4,6), listNames = c("atlas", "sanger"))
getTable(allEqTests, GOLevel = "level 6", onto = "BP")
getTable(allEqTests, GOLevel = "level 6", onto = "BP",
         listNames = c("waldman", "sanger"))
```

`getUpper` *Access to the upper limit of the one-sided confidence intervals for the Sorensen-Dice dissimilarity in one or more equivalence test results*

Description

Given objects representing the result(s) of one or more equivalence tests (classes "equivSDhtest", "equivSDhtestList" or "allEquivSDtest", i.e., the result of functions 'equivTestSorensen' and 'allEquivTestSorensen') this function returns the upper limits of the one-sided confidence intervals [0, dU] for the Sorensen-Dice dissimilarity.

Usage

```
getUpper(x, ...)

## S3 method for class 'equivSDhtest'
getUpper(x, ...)

## S3 method for class 'equivSDhtestList'
getUpper(x, simplify = TRUE, ...)

## S3 method for class 'AllEquivSDhtest'
getUpper(x, onto, GOLevel, listNames, simplify = TRUE, ...)
```

Arguments

x	an object of class "equivSDhtest" or "equivSDhtestList" or "allEquivSDtest".
...	Additional parameters.
simplify	logical, if TRUE the result is simplified, e.g., returning a vector instead of a matrix.
onto	character, a vector with one or more of "BP", "CC" or "MF", ontologies to access.
GOLevel	numeric or character, a vector with one or more GO levels to access. See the details section and the examples.
listNames	character(2), the names of a pair of gene lists.

Details

Argument GOLevel can be of class "character" or "numeric". In the first case, the GO levels must be specified like "level 6" or c("level 4", "level 5", "level 6") In the second case ("numeric"), the GO levels must be specified like6 or seq.int(4,6).

Value

A numeric value, the upper limit of the one-sided confidence interval for the Sorensen-Dice dissimilarity.

When x is an object of class "equivSDhtest" (i.e., the result of a single equivalence test), the returned value is a single numeric value, the upper limit of the one-sided confidence interval for the Sorensen-Dice dissimilarity. For an object of class "equivSDhtestList" (i.e. all pairwise tests for a set of gene lists), if simplify = TRUE (the default), the resulting value is a vector with the upper limit of the one-sided confidence intervals in all those tests, or the symmetric matrix of all these values if simplify = TRUE. If x is an object of class "allEquivSDtest" (i.e., the test iterated along GO ontologies and levels), the preceding result is returned in the form of a list along the ontologies, levels and pairs of gene lists specified by the arguments onto, GOlevel and listNames (or all present in x for missing arguments).

Methods (by class)

- `getUpper(equivSDhtest)`: S3 method for class "equivSDhtest"
- `getUpper(equivSDhtestList)`: S3 method for class "equivSDhtestList"
- `getUpper(AllEquivSDhtest)`: S3 method for class "AllEquivSDhtest"

Examples

```
# Dataset 'allOncoGeneLists' contains the result of the equivalence test between gene lists
# 'sanger' and 'atlas', at level 4 of the BP ontology:
data(eqTest_atlas.sanger_BP4)
eqTest_atlas.sanger_BP4
class(eqTest_atlas.sanger_BP4)
# This may correspond to the result of code like:
# eqTest_atlas.sanger_BP4 <- equivTestSorensen(
#   allOncoGeneLists[["sanger"]], allOncoGeneLists[["atlas"]],
#   geneUniverse = humanEntrezIDs, orgPackg = "org.Hs.eg.db",
#   onto = "BP", GOLevel = 4, listNames = c("sanger", "atlas"))
# (But results may vary according to GO updating)
getUpper(eqTest_atlas.sanger_BP4)

# All pairwise equivalence tests at level 4 of the BP ontology:
data(eqTest_all_BP4)
?eqTest_all_BP4
class(eqTest_all_BP4)
# This may correspond to a call like:
# eqTest_all_BP4 <- equivTestSorensen(allOncoGeneLists,
#                            geneUniverse = humanEntrezIDs, orgPackg = "org.Hs.eg.db",
#                            onto = "BP", GOLevel = 4)
getUpper(eqTest_all_BP4)
getUpper(eqTest_all_BP4, simplify = FALSE)

# Equivalence test iterated over all GO ontologies and levels 3 to 10:
data(allEqTests)
?allEqTests
class(allEqTests)
# This may correspond to code like:
# (By default, the tests are iterated over all GO ontologies and for levels 3 to 10)
# allEqTests <- allEquivTestSorensen(allOncoGeneLists,
#                                             geneUniverse = humanEntrezIDs,
#                                             orgPackg = "org.Hs.eg.db")
# All upper confidence limits for the Sorensen-Dice dissimilarities:
getUpper(allEqTests)
getUpper(allEqTests, simplify = FALSE)

# Upper confidence limits only for some GO ontologies, levels or pairs of gene lists:
getUpper(allEqTests, GOLevel = "level 6")
getUpper(allEqTests, GOLevel = 6)
getUpper(allEqTests, GOLevel = seq.int(4,6))
getUpper(allEqTests, GOLevel = "level 6", simplify = FALSE)
getUpper(allEqTests, GOLevel = "level 6", listNames = c("atlas", "sanger"))
getUpper(allEqTests, GOLevel = seq.int(4,6), onto = "BP")
```

```
getUpper(allEqTests, GOLevel = seq.int(4,6), onto = "BP", simplify = FALSE)
getUpper(allEqTests, GOLevel = "level 6", onto = "BP",
         listNames = c("waldman", "sanger"))
getUpper(allEqTests$BP$`level 4`)
```

---

gosorensen *gosorensen: A package for making inference on gene lists based on the Sorensen-Dice dissimilarity*

Description

Given two lists of genes, and a set of Gene Ontology (GO) items (e.g., all GO items in a given level of a given GO ontology) one may explore some aspects of their biological meaning by constructing a 2x2 contingency table, the cross-tabulation of: number of these GO items non-enriched in both gene lists (n00), items enriched in the first list but not in the second one (n10), items non-enriched in the first list but enriched in the second (n10) and items enriched in both lists (n11). Then, one may express the degree of similarity or dissimilarity between the two lists by means of an appropriate index computed on these frequency tables of concordance or non-concordance in GO items enrichment. In our opinion, an appropriate index is the Sorensen-Dice index which ignores the double negatives n00: if the total number of candidate GO items under consideration grows (e.g., all items in a deep level of an ontology) likely n00 will also grow artificially. On the other hand, intuitively the degree of similarity between both lists must be directly related to the degree of concordance in the enrichment, n11.

Details

gosorensen package provides the following functions:

enrichedIn Build a cross-tabulation of enriched and non-enriched GO terms vs. gene lists

buildEnrichTable Build an enrichment contingency table from two or more gene lists

allBuildEnrichTable Iterate 'buildEnrichTable' along the specified GO ontologies and GO levels

nice2x2Table Check for validity an enrichment contingency table

dSorensen Compute the Sorensen-Dice dissimilarity

seSorensen Standard error estimate of the sample Sorensen-Dice dissimilarity

duppSorensen Upper limit of a one-sided confidence interval (0,dUpp] for the population dissimilarity

equivTestSorensen Equivalence test between two gene lists, based on the Sorensen-Dice dissimilarity

allEquivTestSorensen Iterate equivTestSorensen along GO ontologies and GO levels

getDissimilarity, getPvalue, getSE, getTable, getUpper, getNboot, getEffNboot Accessor functions to some fields of an equivalence test result

upgrade Updating the result of an equivalence test, e.g., changing the equivalence limit

sorenThreshold For a given level (2, 3, ...) in a GO ontology (BP, MF or CC), compute the equivalence threshold dissimilarity matrix.

allSorenThreshold Iterate 'sorenThreshold' along the specified GO ontologies and GO levels.

hclustThreshold From a Sorensen-Dice threshold dissimilarity matrix, generate an object of class "hclust"

allHclustThreshold Iterate 'hclustThreshold' along the specified GO ontologies and GO levels

pruneClusts Remove all NULL or unrepresentable as a dendrogram "equivClustSorensen" elements in an object of class "equivClustSorensenList"

All these functions are generic, adequate for different (S3) classes representing the before cited GO term enrichment cross-tabulations.

Author(s)

Maintainer: Pablo Flores <p_flores@espoch.edu.ec> (ORCID)

Authors:

- Jordi Ocana (ORCID) [contributor]

Other contributors:

- Alexandre Sanchez-Pla (ORCID) [contributor]
- Miquel Salicru (ORCID) [contributor]

`hclustThreshold` *From a Sorensen-Dice threshold dissimilarity matrix, generate an object of class "hclust"*

Description

From a Sorensen-Dice threshold dissimilarity matrix, generate an object of class "hclust"

Usage

```
hclustThreshold(
  x,
  onTheFlyDev = NULL,
  method = "complete",
  jobName = paste("Equivalence cluster", method, sep = "_"),
  ylab = "Sorensen equivalence threshold dissimilarity",
  ...
)
```

Arguments

x	an object of class "dist" with the Sorensen-Dice equivalence threshold dissimilarities matrix
onTheFlyDev	character, name of the graphical device where to immediately display the resulting diagram. The appropriate names depend on the operating system. Defaults to NULL and then nothing is displayed
method	character, one of the admissible methods in function hclust. Defaults to "complete"
jobName	character, main plot name, defaults to paste("Equivalence cluster", onto, ontoLevel, method, sep = "_")
ylab	character, label of the vertical axis of the plot, defaults to "Sorensen equivalence threshold dissimilarity"
...	additional arguments to hclust

Value

An object of class equivClustSorensen, descending from class hclust

Examples

```
# Gene lists to analyse:
data("allOncoGeneLists")

# Obtaining ENTREZ identifiers for the gene universe of humans:
library(org.Hs.eg.db)
humanEntrezIDs <- keys(org.Hs.eg.db, keytype = "ENTREZID")

# First, compute the Sorensen-Dice threshold equivalence dissimilarity
# for ontology BP at level 4:
# # Very time consuming, it requires building all joint enrichment contingency tables
# dOncBP4 <- sorenThreshold(allOncoGeneLists, onto = "BP", GOLevel = 4,
#                           geneUniverse = humanEntrezIDs, orgPackg = "org.Hs.eg.db")
# Better (much faster), using the previously tabulated contingency tables:
data("cont_all_BP4")
dOncBP4 <- sorenThreshold(cont_all_BP4)
clust.threshold <- hclustThreshold(dOncBP4)
plot(clust.threshold, main = "AllOnco genelists, BP ontology at level 4",
     ylab = "Sorensen equivalence threshold")
# With the same data, an UPGMA dendrogram:
clust.threshold <- hclustThreshold(dOncBP4, method = "average")
plot(clust.threshold, main = "AllOnco genelists, BP ontology at level 4",
     ylab = "Sorensen equivalence threshold")
```

`nice2x2Table` *Checks for validity data representing an enrichment contingency table generated from two gene lists*

Description

Checks for validity data representing an enrichment contingency table generated from two gene lists

Usage

```
nice2x2Table(x)

## S3 method for class 'table'
nice2x2Table(x)

## S3 method for class 'matrix'
nice2x2Table(x)

## S3 method for class 'numeric'
nice2x2Table(x)
```

Arguments

x	either an object of class "table", "matrix" or "numeric".

Details

In the "table" and "matrix" interfaces, the input parameter x must correspond to a two-dimensional array:

$$\begin{matrix} n_{11} & n_{10} \\ n_{01} & n_{00}, \end{matrix}$$

These values are interpreted (always in this order) as n11: number of GO terms enriched in both lists, n01: GO terms enriched in the second list but not in the first one, n10: terms not enriched in the second list but enriched in the first one and double negatives, n00. The double negatives n00 are ignored in many computations concerning the Sorensen-Dice index.

In the "numeric" interface, the input x must correspond to a numeric of length 3 or more, in the same order as before.

Value

boolean, TRUE if x nicely represents a 2x2 contingency table interpretable as the cross-tabulation of the enriched GO terms in two gene lists: "Number of enriched terms in list 1 (TRUE, FALSE)" x "Number of enriched terms in list 2 (TRUE, FALSE)". In this function, "nicely representing a 2x2 contingency table" is interpreted in terms of computing the Sorensen-Dice dissimilarity and associated statistics. Otherwise the execution is interrupted.

Methods (by class)

- nice2x2Table(table): S3 method for class "table"
- nice2x2Table(matrix): S3 method for class "matrix"
- nice2x2Table(numeric): S3 method for class "numeric"

Examples

```
conti <- as.table(matrix(c(27, 36, 12, 501, 43, 15, 0, 0, 0), nrow = 3, ncol = 3,
                         dimnames = list(c("a1","a2","a3"),
                                         c("b1", "b2","b3"))))
tryCatch(nice2x2Table(conti), error = function(e) {return(e)})
conti2 <- conti[1,seq.int(1, min(2,ncol(conti))), drop = FALSE]
conti2
tryCatch(nice2x2Table(conti2), error = function(e) {return(e)})

conti3 <- matrix(c(12, 210), ncol = 2, nrow = 1)
conti3
tryCatch(nice2x2Table(conti3), error = function(e) {return(e)})

conti4 <- c(32, 21, 81, 1439)
nice2x2Table(conti4)
conti4.mat <- matrix(conti4, nrow = 2)
conti4.mat
conti5 <- c(32, 21, 81)
nice2x2Table(conti5)

conti6 <- c(-12, 21, 8)
tryCatch(nice2x2Table(conti6), error = function(e) {return(e)})

conti7 <- c(0, 0, 0, 32)
tryCatch(nice2x2Table(conti7), error = function(e) {return(e)})
```

---

pbtGeneLists *14 gene lists possibly related with kidney transplant rejection*

Description

An object of class "list" of length 14. A non up-to-date subset of the University of Alberta pathogenesis-based transcripts sets (PBTs) that were generated by using Affymetrix Microarrays. Take them just as an illustrative example.

Usage

```
data(pbtGeneLists)
```

Format

An object of class "list" of length 5. Each one of its elements is a "character" vector of ENTREZ gene identifiers.

Source

https://www.ualberta.ca/medicine/institutes-centres-groups/atagc/research/gene-lists.html

---

`pruneClusts` *Remove all NULL or unrepresentable as a dendrogram "equivClustSorensen" elements in an object of class "equivClustSorensenList"*

Description

Remove all NULL or unrepresentable as a dendrogram "equivClustSorensen" elements in an object of class "equivClustSorensenList"

Usage

```
pruneClusts(x)
```

Arguments

`x`	An object of class "equivClustSorensenList" descending from "iterEquivClust" which itself descends from class "list". See the details section.

Details

"equivClustSorensenList" objects are lists whose components are one or more of BP, CC or MF, the GO ontologies. Each of these elements is itself a list whose elements correspond to GO levels. Finally, the elements of these lists are objects of class "equivClustSorensen", descending from "equivClust" which itself descends from "hclust".

Value

An object of class "equivClustSorensenList".

`seSorensen` *Standard error of the sample Sorensen-Dice dissimilarity, asymptotic approach*

Description

Standard error of the sample Sorensen-Dice dissimilarity, asymptotic approach

Usage

```
seSorensen(x, ...)

## S3 method for class 'table'
seSorensen(x, check.table = TRUE, ...)

## S3 method for class 'matrix'
seSorensen(x, check.table = TRUE, ...)

## S3 method for class 'numeric'
seSorensen(x, check.table = TRUE, ...)

## S3 method for class 'character'
seSorensen(x, y, check.table = TRUE, ...)

## S3 method for class 'list'
seSorensen(x, check.table = TRUE, ...)

## S3 method for class 'tableList'
seSorensen(x, check.table = TRUE, ...)
```

Arguments

x	either an object of class "table", "matrix" or "numeric" representing a 2x2 contingency table, or a "character" (a set of gene identifiers) or "list" or "tableList" object. See the details section for more information.
...	extra parameters for function buildEnrichTable.
check.table	Boolean. If TRUE (default), argument x is checked to adequately represent a 2x2 contingency table. This checking is performed by means of function nice2x2Table.
y	an object of class "character" representing a vector of gene identifiers (e.g., ENTREZ).

Details

This function computes the standard error estimate of the sample Sorensen-Dice dissimilarity, given a 2x2 arrangement of frequencies (either implemented as a "table", a "matrix" or a "numeric" object):

$$\begin{matrix} n_{11} & n_{10} \\ n_{01} & n_{00}, \end{matrix}$$

The subindex '11' corresponds to those GO terms enriched in both lists, '01' to terms enriched in the second list but not in the first one, '10' to terms enriched in the first list but not enriched in the second one and '00' corresponds to those GO terms non enriched in both gene lists, i.e., to the double negatives, a value which is ignored in the computations.

In the "numeric" interface, if `length(x) >= 3`, the values are interpreted as (n_{11}, n_{01}, n_{10}), always in this order.

If `x` is an object of class "character", then `x` (and `y`) must represent two "character" vectors of valid gene identifiers (e.g., ENTREZ). Then the standard error for the dissimilarity between lists `x` and `y` is computed, after internally summarizing them as a 2x2 contingency table of joint enrichment. This last operation is performed by function `buildEnrichTable` and "valid gene identifiers (e.g., ENTREZ)" stands for the coherency of these gene identifiers with the arguments `geneUniverse` and `orgPackg` of `buildEnrichTable`, passed by the ellipsis argument `...` in `seSorensen`.

In the "list" interface, the argument must be a list of "character" vectors, each one representing a gene list (character identifiers). Then, all pairwise standard errors of the dissimilarity between these gene lists are computed.

If `x` is an object of class "tableList", the standard error of the Sorensen-Dice dissimilarity estimate is computed over each one of these tables. Given k gene lists (i.e. "character" vectors of gene identifiers) l1, l2, ..., lk, an object of class "tableList" (typically constructed by a call to function `buildEnrichTable`) is a list of lists of contingency tables t(i,j) generated from each pair of gene lists i and j, with the following structure:

$l2

$l2$l1$t(2,1)

$l3

$l3$l1$t(3,1), $l3$l2$t(3,2)

...

$lk

lkl1$t(k,1), lkl2$t(k,2), ..., lkl(k-1)t(k,k-1)

Value

In the "table", "matrix", "numeric" and "character" interfaces, the value of the standard error of the Sorensen-Dice dissimilarity estimate. In the "list" and "tableList" interfaces, the symmetric matrix of all standard error dissimilarity estimates.

Methods (by class)

- `seSorensen(table)`: S3 method for class "table"
- `seSorensen(matrix)`: S3 method for class "matrix"
- `seSorensen(numeric)`: S3 method for class "numeric"
- `seSorensen(character)`: S3 method for class "character"
- `seSorensen(list)`: S3 method for class "list"
- `seSorensen(tableList)`: S3 method for class "tableList"

See Also

`buildEnrichTable` for constructing contingency tables of mutual enrichment, `nice2x2Table` for checking the validity of enrichment contingency tables, `dSorensen` for computing the Sorensen-Dice dissimilarity, `duppSorensen` for the upper limit of a one-sided confidence interval of the dissimilarity, `equivTestSorensen` for an equivalence test.

Examples

```
# Gene lists 'atlas' and 'sanger' in 'allOncoGeneLists' dataset. Table of joint enrichment
# of GO terms in ontology BP at level 4.
data(cont_atlas.sanger_BP4)
cont_atlas.sanger_BP4
dSorensen(cont_atlas.sanger_BP4)
seSorensen(cont_atlas.sanger_BP4)

# Contingency table as a numeric vector:
seSorensen(c(56, 1, 30, 47))
seSorensen(c(56, 1, 30))

# (These examples may be considerably time consuming due to many enrichment
# tests to build the contingency tables of mutual enrichment)
# data(allOncoGeneLists)
# ?allOncoGeneLists

# Standard error of the sample Sorensen-Dice dissimilarity, directly from
# two gene lists, from scratch:
# seSorensen(allOncoGeneLists$atlas, allOncoGeneLists$sanger,
#            onto = "BP", GOLevel = 3,
#            geneUniverse = humanEntrezIDs, orgPackg = "org.Hs.eg.db")
# Essentially, the above code makes the same as:
# cont_atlas.sanger_BP4 <- buildEnrichTable(allOncoGeneLists$atlas, allOncoGeneLists$sanger,
#                                   onto = "BP", GOLevel = 4,
#                               geneUniverse = humanEntrezIDs, orgPackg = "org.Hs.eg.db")
# cont_atlas.sanger_BP4
# seSorensen(cont_atlas.sanger_BP4)

# All pairwise standard errors (quite time consuming):
# seSorensen(allOncoGeneLists,
#            onto = "BP", GOLevel = 4,
#            geneUniverse = humanEntrezIDs, orgPackg = "org.Hs.eg.db")
```

---

sorenThreshold *For a given level (2, 3, ...) in a GO ontology (BP, MF or CC), compute the equivalence threshold dissimilarity matrix.*

Description

For a given level (2, 3, ...) in a GO ontology (BP, MF or CC), compute the equivalence threshold dissimilarity matrix.

Usage

```
sorenThreshold(x, ...)

## S3 method for class 'list'
sorenThreshold(
```

```
  x,
  onto,
  GOLevel,
  geneUniverse,
  orgPackg,
  boot = FALSE,
  nboot = 10000,
  boot.seed = 6551,
  trace = TRUE,
  alpha = 0.05,
  precis = 0.001,
  ...
)

## S3 method for class 'tableList'
sorenThreshold(
  x,
  boot = FALSE,
  nboot = 10000,
  boot.seed = 6551,
  trace = TRUE,
  alpha = 0.05,
  precis = 0.001,
  ...
)
```

Arguments

x	either an object of class "list" or class "tableList". See the details section for more information.
...	additional arguments to buildEnrichTable
onto	character, GO ontology ("BP", "MF" or "CC") under consideration
GOLevel	integer (2, 3, ...) level of a GO ontology where the GO profiles are built
geneUniverse	character vector containing the universe of genes from where geneLists have been extracted. This vector must be extracted from the annotation package declared in orgPackg. For more details see README File.
orgPackg	A string with the name of the genomic annotation package corresponding to a specific species to be analyzed, which must be previously installed and activated. For more details see README File.
boot	boolean. If TRUE, the p-values are computed by means of a bootstrap approach instead of the asymptotic normal approach. Defaults to FALSE.
nboot	numeric, number of initially planned bootstrap replicates. Ignored if boot == FALSE. Defaults to 10000
boot.seed	starting random seed for all bootstrap iterations. Defaults to 6551. see the details section
trace	boolean, the full process must be traced? Defaults to TRUE

alpha	simultaneous nominal significance level for the equivalence tests to be repeteadly performed, defaults to 0.05
precis	numerical precision in the iterative search of the equivalence threshold dissimilarities, defaults to 0.001

Details

If x is an object of class "list", each of its elements must be a "character" vector of gene identifiers (e.g., ENTREZ). Then all pairwise threshold dissimilarities between these gene lists are obtained.

Class "tableList" corresponds to objects representing all mutual enrichment contingency tables generated in a pairwise fashion: Given gene lists l1, l2, ..., lk, an object of class "tableList" (typically constructed by a call to function buildEnrichTable) is a list of lists of contingency tables tij generated from each pair of gene lists i and j, with the following structure:

$l2

$l2$l1$t21

$l3

$l3$l1$t31, $l3$l2$t32

...

lkl1$tk1, lkl2$tk2, ..., lkl(k-1)tk(k-1)

If x is an object of class "tableList", the threshold dissimilarity is obtained over each one of these tables.

If boot == TRUE, all series of nboot bootstrap replicates start from the same random seed, provided by the argument boot.seed, except if boot == NULL.

Do not confuse the resulting threshold dissimilarity matrix with the Sorensen-Dice dissimilarities computed in each equivalence test.

The dimension of the resulting matrix may be less than the number of original gene lists being compared, as the process may not converge for some pairs of gene lists.

Value

An object of class "dist", the equivalence threshold dissimilarity matrix based on the Sorensen-Dice dissimilarity.

Methods (by class)

- sorenThreshold(list): S3 method for class "list"
- sorenThreshold(tableList): S3 method for class "tableList"

Examples

```
# Gene lists to be explored for enrichment:
data(allOncoGeneLists)

# Obtaining ENTREZ identifiers for the gene universe of humans:
library(org.Hs.eg.db)
humanEntrezIDs <- keys(org.Hs.eg.db, keytype = "ENTREZID")
```

```
# This example is quite time consuming:
# sorenThreshold(allOncoGeneLists,
#                geneUniverse = humanEntrezIDs, orgPackg = "org.Hs.eg.db")
# Much faster:
# Object \code{cont_all_BP4} of class "tableList" contains all the pairwise contingency
# tables of joint enrichment for the gene lists in \code{allOncoGeneLists}, for the BP
# GO ontology at level 4:
data("cont_all_BP4")
sorenThreshold(cont_all_BP4)
```

---

upgrade *Update the result of a Sorensen-Dice equivalence test.*

Description

Recompute the test (or tests) from an object of class "equivSDhtest", "equivSDhtestList" or "AllEquivSDhtest" (i.e.,the output of functions "equivTestSorensen" or "allEquivTestSorensen"). Using the same table or tables of enrichment frequencies in 'x', obtain again the result of the equivalence test for new values of any of the parameters `d0` or `conf.level` or `boot` or `nboot` or `check.table`.

Usage

```
upgrade(x, ...)

## S3 method for class 'equivSDhtest'
upgrade(x, ...)

## S3 method for class 'equivSDhtestList'
upgrade(x, ...)

## S3 method for class 'AllEquivSDhtest'
upgrade(x, ...)
```

Arguments

`x`	an object of class "equivSDhtest", "equivSDhtestList" or "AllEquivSDhtest".
`...`	any valid parameters for function "equivTestSorensen" for its interface "table", to recompute the test(s) according to these parameters.

Value

An object of the same class than `x`.

Methods (by class)

- `upgrade(equivSDhtest)`: S3 method for class "equivSDhtest"
- `upgrade(equivSDhtestList)`: S3 method for class "equivSDhtestList"
- `upgrade(AllEquivSDhtest)`: S3 method for class "allEquivSDhtest"

Examples

```
# Result of the equivalence test between gene lists 'sanger' and 'atlas', in dataset
# 'allOncoGeneLists', at level 4 of the BP ontology:
data(eqTest_atlas.sanger_BP4)
eqTest_atlas.sanger_BP4
class(eqTest_atlas.sanger_BP4)
# This may correspond to the result of code like:
# data(allOncoGeneLists)
# library(org.Hs.eg.db)
# humanEntrezIDs <- keys(org.Hs.eg.db, keytype = "ENTREZID")
# eqTest_atlas.sanger_BP4 <- equivTestSorensen(
#   allOncoGeneLists[["sanger"]], allOncoGeneLists[["atlas"]],
#   geneUniverse = humanEntrezIDs, orgPackg = "org.Hs.eg.db",
#   onto = "BP", GOLevel = 4, listNames = c("sanger", "atlas"))
upgrade(eqTest_atlas.sanger_BP4, d0 = 1/(1 + 10/9)) # d0 = 0.4737
upgrade(eqTest_atlas.sanger_BP4, d0 = 1/(1 + 2*1.25)) # d0 = 0.2857
upgrade(eqTest_atlas.sanger_BP4, d0 = 1/(1 + 2*1.25), conf.level = 0.99)

# All pairwise equivalence tests at level 4 of the BP ontology
data(eqTest_all_BP4)
?eqTest_all_BP4
class(eqTest_all_BP4)
# This may correspond to a call like:
# data(allOncoGeneLists)
# library(org.Hs.eg.db)
# humanEntrezIDs <- keys(org.Hs.eg.db, keytype = "ENTREZID")
# eqTest_all_BP4 <- equivTestSorensen(allOncoGeneLists,
#                           geneUniverse = humanEntrezIDs, orgPackg = "org.Hs.eg.db",
#                           onto = "BP", GOLevel = 4)
upgrade(eqTest_all_BP4, d0 = 1/(1 + 2*1.25)) # d0 = 0.2857

data(allEqTests)
?allEqTests
class(allEqTests)
upgrade(allEqTests, d0 = 1/(1 + 2*1.25)) # d0 = 0.2857
```

Index

∗ **datasets**
allContTabs, 5
allDissMatrx, 5
allEqTests, 6
allEqTests_boot, 6
allOncoGeneLists, 10
cont_all_BP4, 17
cont_atlas.sanger_BP4, 18
dissMatrx_BP4, 19
enrichedInBP4, 30
eqTest_all_BP4, 31
eqTest_atlas.sanger_BP4, 32
fullEnrichedInBP4, 37
pbtGeneLists, 59

allBuildEnrichTable, 3
allContTabs, 5
allDissMatrx, 5
allEqTests, 6
allEqTests_boot, 6
allEquivTestSorensen, 7
allHclustThreshold, 9
allOncoGeneLists, *5–7*, 10, *18, 19*, *30–32*, *37, 38*
allSorenThreshold, 11

boot.tStat, 13
buildEnrichTable, 14, *20, 21*, *25, 26*, *35*, *62*, *65*

cont_all_BP4, 17
cont_atlas.sanger_BP4, 18

dissMatrx_BP4, 19
dSorensen, 19, *26*, *36*, *62*
duppSorensen, *21*, 22, *36*, *62*

enrichedIn, *18*, 27, *30*, *38*
enrichedInBP4, 30
eqTest_all_BP4, 31
eqTest_atlas.sanger_BP4, 32
equivTestSorensen, *21*, *26*, 32, *62*

fullEnrichedInBP4, 37

getDissimilarity, 38
getEffNboot, *36*, 40, *44*
getNboot, *36*, *42*, 43
getPvalue, *36*, 46
getSE, *36*, 48
getTable, *36*, 50
getUpper, *36*, 52
goSorensen *(*gosorensen*)*, 55
gosorensen, 55
goSorensen-package *(*gosorensen*)*, 55

hclustThreshold, 56

nice2x2Table, *21*, *26*, *36*, 58, *62*

pbtGeneLists, 59
pruneClusts, 60

seSorensen, *21*, *26*, *36*, 60
sorenThreshold, 63

update, *36*
upgrade, 66