Package 'goSorensen'

March 12, 2023

Type Package

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

Version 1.0.0

Description This package implements inferential methods to compare gene lists (in this first release, to prove equivalence) 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.

Suggests BiocStyle, knitr, rmarkdown

Depends R (>= 4.2.0)

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

VignetteBuilder knitr

License GPL-3

Encoding UTF-8

LazyData false

RoxygenNote 7.2.0

NeedsCompilation no

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

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

git_branch RELEASE_3_16

git_last_commit a78ba7a

git_last_commit_date 2022-11-01

Date/Publication 2023-03-12

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>

R topics documented:

`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,
  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),
  ...
)
```

Arguments

`x`	object of class "list". Each of its elements must be a "character" vector of gene identifiers. Then all pairwise equivalence tests are performed between these gene lists, iterating the process for all specified GO ontologies and GO levels.
`d0`	equivalence threshold for the population Sorensen-Dice dissimilarity, d. The null hypothesis states that d >= d0, and the alternative that d < 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 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`.
`ontos`	"character", GO ontologies to analyse. Defaults to `c("BP", "CC", "MF")`.
`GOLevels`	"integer", GO levels to analyse inside each of these GO ontologies.
`...`	extra parameters for function `buildEnrichTable`.

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 all pairwise comparisons between the gene lists in argument `x`.

Examples

```
# This example is extremely time consuming, it scans two GO ontologies and three
# GO levels inside them to perform the equivalence test.
# Gene universe:
# data(humanEntrezIDs)
# Gene lists to be explored for enrichment:
# data(pbtGeneLists)
# allEquivTestSorensen(pbtGeneLists,
#                      geneUniverse = humanEntrezIDs, orgPackg = "org.Hs.eg.db",
#                      ontos = c("MF", "BP"), GOLevels = seq.int(4,6))
```

---

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. 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 gene identifiers.

Source

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

`allTabsBP.4` *An example of "tableList" object resulting from a call to 'buildEnrichTable'*

Description

The result of generating all contingency tables of mutual enrichment, in a pairwise fashion, between the gene lists in data `allOncoGeneLists`. The information in these data was summarized as 2x2 contingency tables of GO items enrichment, at level 4 of the BP ontology. These results are based on gene lists which are non automatically updated, take them just as an illustrative example because the gene lists, and the GO, may change along time.

Usage

```
data(allTabsBP.4)
```

Format

An object of class "tableList" inheriting from class "list". It is a list of class "table" objects.

`boot.cancerEquivSorensen` *An example of "AllEquivSDhtest" object resulting from a call to 'allEquivTestSorensen'*

Description

The bootstrap Sorensen-Dice test performed on the cancer gene lists in data `allOncoGeneLists` which is automatically charged with this package. The test is iterated for all GO ontologies and for GO levels 3 to 10. These results are not automatically updated for changes in these gene lists and Bioconductor and Go updates, take them just as an illustrative example.

Usage

```
data(boot.cancerEquivSorensen)
```

Format

An object of class "AllEquivSDhtest" inheriting from class "list". Each one of its elements, named BP, CC and MF respectively, corresponds to a GO ontology. It is itself a list of length 8 whose elements are named as "Level 3" to "Level 10". For each combination of ontology and level, there is an object of class "equivSDhtestList" codifying the result of all pairwise tests between these cancer gene lists.

Details

For each ontology and GO level, the result contains the result of all pairwise tests of equivalence between the cancer gene lists.

Source

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

`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):

n_11 n_01
n_10 n_00,

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

`BP.4` *An example of "equivSDhtestList" object resulting from a call to 'equivSorensenTest'*

Description

The result of all pairwise Sorensen-Dice equivalence tests between the gene lists in data `allOncoGeneLists` which is automatically charged with this package. To perform the tests, the information in these data was summarized as 2x2 contingency tables of GO items enrichment, at level 4 of the BP ontology, and the tests were performed for an equivalence limit d0 = 0.4444 and a confidence level conf.int = 0.95. These results are based on gene lists which are non automatically updated, take them just as an illustrative example.

Usage

```
data(BP.4)
```

Format

An object of class "equivSDhtestList" inheriting from class "list". It is a list of class "equivSDhtest" objects.

Source

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

`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,
  restricted = FALSE,
  pAdjustMeth = "BH",
  pvalCutoff = 0.01,
  qvalCutoff = 0.05,
  ...
)

## S3 method for class 'character'
buildEnrichTable(
  x,
  y,
  listNames = c("gene.list1", "gene.list2"),
  check.table = TRUE,
  geneUniverse,
  orgPackg,
  onto,
  GOLevel,
  restricted = FALSE,
  pAdjustMeth = "BH",
  pvalCutoff = 0.01,
  qvalCutoff = 0.05,
  ...
)

## S3 method for class 'list'
buildEnrichTable(
  x,
  check.table = TRUE,
  geneUniverse,
  orgPackg,
  onto,
  GOLevel,
  restricted = FALSE,
  pAdjustMeth = "BH",
  pvalCutoff = 0.01,
```

```
  qvalCutoff = 0.05,
  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 or an object of class "list". In this second case, each element of the list must be a "character" vector of gene identifiers. 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.
listNames	a character(2) with the gene lists names originating the cross-tabulated enrichment frequencies.
check.table	Logical The resulting table must be checked. Defaults to TRUE.
geneUniverse	character vector containing all genes from where geneLists have been extracted.
orgPackg	A string with the name of the annotation package.
onto	string describing the ontology. Either "BP", "MF" or "CC".
GOLevel	An integer, the GO ontology level.
restricted	Logical variable to decide how tabulation of GOIDs is performed. Defaults to FALSE. See the details section.
pAdjustMeth	string describing the adjust method, either "BH", "BY" or "Bonf", defaults to 'BH'.
pvalCutoff	A numeric value. Defaults to 0.01.
qvalCutoff	A numeric value. Defaults to 0.05.
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

Unrestricted tabulation crosses _all_ GO Terms located at the level indicated by 'GOLev' with the two GOIDs lists. Restricted tabulation crosses only terms from the selected GO level that are _common to ancestor terms of either list_. That is, if one term in the selected GO level is not an ancestor of at least one of the gene list most specific GO terms it is excluded from the GO Level's terms because it is impossible that it appears as being enriched.

Value

in the "character" interface, an object of class "table" is returned. It represents a 2x2 contingency table interpretable as the cross-tabulation of the enriched GO items in two gene lists: "Number of enriched items in list 1 (TRUE, FALSE)" x "Number of enriched items 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)

- `default`: S3 default method
- `character`: S3 method for class "character"
- `list`: S3 method for class "list"

Examples

```
# Gene universe:
data(humanEntrezIDs)
# Gene lists to be explored for enrichment:
data(allOncoGeneLists)
?allOncoGeneLists
# Table of mutual GO node enrichment between gene lists Vogelstein and sanger,
# for ontology MF at GO level 6 (only first 50 genes, to improve speed).
vog.VS.sang <- buildEnrichTable(allOncoGeneLists[["Vogelstein"]][seq_len(50)],
                                allOncoGeneLists[["sanger"]][seq_len(50)],
                               geneUniverse = humanEntrezIDs, orgPackg = "org.Hs.eg.db",
                          onto = "MF", GOLevel = 6, listNames = c("Vogelstein", "sanger"))
vog.VS.sang
# This is an inadequate table for Sorensen-Dice computations:
equivTestSorensen(vog.VS.sang)
# This sometimes happens, due too small gene lists or due to poor incidence
# of enrichment.
#
# In fact, the complete gene lists generate a much interesting contingency table:
# 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
# equivTestSorensen(vog.VS.sang)
```

`cancerEquivSorensen` *An example of "AllEquivSDhtest" object resulting from a call to 'allEquivTestSorensen'*

Description

The Sorensen-Dice test (normal asymptotic version) performed on the cancer gene lists in data `allOncoGeneLists` which is automatically charged with this package. The test is iterated for all GO ontologies and for GO levels 3 to 10. These results are not automatically updated for changes in these gene lists and Bioconductor or Go updates, take them just as an illustrative example.

Usage

```
data(cancerEquivSorensen)
```

Format

An object of class "AllEquivSDhtest" inheriting from class "list". Each one of its elements, named BP, CC and MF respectively, corresponds to a GO ontology. It is itself a list of length 8 whose elements are named as "Level 3" to "Level 10". For each combination of ontology and level, there is an object of class "equivSDhtestList" codifying the result of all pairwise tests between these cancer gene lists.

Details

For each ontology and GO level, the result contains the result of all pairwise tests of equivalence between the cancer gene lists.

Source

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

`completeTable` *Reformats and completes (if necessary) an enrichment contingency table as is generated by function 'crossTabGOIDs4GeneLists', in order to make it appropriate for its use in package goSorensen.*

Description

Reformats and completes (if necessary) an enrichment contingency table as is generated by function 'crossTabGOIDs4GeneLists', in order to make it appropriate for its use in package goSorensen.

Usage

```
completeTable(x, listNames)
```

Arguments

x	an object of class "table", typically the output of function 'crossTabGOIDs4GeneLists'.
listNames	a character(2) with the gene lists names originating the cross-tabulated enrichment frequencies.

Value

a complete contingency table to use in package goSorensen.

---

crossTabGOIDs *crossTabGOIDs*

Description

This function performs a crosstabulation between two lists of enriched GOTerms The lists are intended to have been obtained from enrichment analyses performed on two gene lists

Usage

```
crossTabGOIDs(
  GO1,
  GO2,
  onto,
  GOLev,
  listNames = NULL,
  geneList1 = NULL,
  geneList2 = NULL,
  orgPackage = NULL,
  restricted = FALSE
)
```

Arguments

GO1	character vector containing a FIRST list of GO identifiers
GO2	character vector containing a SECOND gene list GO identifiers
onto	string describing the ontology. Belongs to c('BP', 'MF', 'CC', 'ANY')
GOLev	An integer
listNames	character vector with names of the genelists that generated the enriched GOIDs
geneList1	character vector containing a FIRST gene list of entrez IDs
geneList2	character vector containing a SECOND gene list of entrez IDs
orgPackage	A string wih the name of the annotation package

restricted	Boolean variable to decide how tabulation is performed. Unrestricted tabulation crosses _all_ GO Terms located at the level indicated by 'GOLev' with the two GOIDs lists Restricted tabulation crosses only terms from the selected GO level that are _common to ancestor terms of either list_. That is, if one term in the selected GO level is not an ancestor of at least one of the gene list most specific GO terms it is excluded from the GO Level's terms because it is impossible that it appears as being enriched.

Value

performs a crosstabulation between two lists of enriched GOTerms

---

`crossTabGOIDs4GeneLists` *crossTab4GeneLists*

Description

This function builds a cross-tabulation of enriched and non-enriched GO terms from two gene lists

Usage

```
crossTabGOIDs4GeneLists(
  genelist1,
  genelist2,
  geneUniverse,
  orgPackg,
  onto,
  GOLev,
  restricted = FALSE,
  pAdjustMeth = "BH",
  pvalCutoff = 0.01,
  qvalCutoff = 0.05
)
```

Arguments

genelist1	character vector containing a FIRST gene list of entrez IDs
genelist2	character vector containing a SECOND gene list of entrez IDs
geneUniverse	character vector containing all genes from where geneLists have been extracted
orgPackg	A string wih the name of the annotation package
onto	string describing the ontology. Belongs to c('BP', 'MF', 'CC', 'ANY')
GOLev	An integer

restricted	Boolean variable to decide how tabulation of GOIDs is performed. Unrestricted tabulation crosses _all_ GO Terms located at the level indicated by 'GOLev' with the two GOIDs lists Restricted tabulation crosses only terms from the selected GO level that are _common to ancestor terms of either list_. That is, if one term in the selected GO level is not an ancestor of at least one of the gene list most specific GO terms it is excluded from the GO Level's terms because it is impossible that it appears as being enriched.
pAdjustMeth	string describing the adjust method. Belongs to c('BH', 'BY', 'Bonf')
pvalCutoff	A numeric value
qvalCutoff	A numeric value

Value

a cross-tabulation of enriched and non-enriched GO terms from two gene lists

---

`crossTabGOIDsUnrestricted`

crossTabGOIDsUnrestricted

Description

This function performs a crosstabulation between two lists of enriched GOTerms The lists are intended to have been obtained from enrichment analyses performed on two gene lists

Usage

```
crossTabGOIDsUnrestricted(GO1, GO2, onto, GOLev, listNames = NULL)
```

Arguments

GO1	character vector containing a FIRST list of GO identifiers
GO2	character vector containing a SECOND gene list GO identifiers
onto	string describing the ontology. Belongs to c('BP', 'MF', 'CC', 'ANY')
GOLev	An integer
listNames	character vector with names of the genelists that generated the enriched GOIDs

Value

a crosstabulation between two lists of enriched GOTerms

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.

Details

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

$$\begin{matrix} n11 & n01 \\ n10 & n00, \end{matrix}$$

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 items enriched in both lists, '01' to items enriched in the second list but not in the first one, '10' to items enriched in the first list but not enriched in the second one and '00' corresponds to those GO items 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. 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" 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)

- `table`: S3 method for class "table"
- `matrix`: S3 method for class "matrix"
- `numeric`: S3 method for class "numeric"
- `character`: S3 method for class "character"
- `list`: S3 method for class "list"
- `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 items in ontology BP at level 3.
data(tab_atlas.sanger_BP3)
tab_atlas.sanger_BP3
?tab_atlas.sanger_BP3
dSorensen(tab_atlas.sanger_BP3)

# 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 mutual enrichment)
# data(pbtGeneLists)
# ?pbtGeneLists
# data(humanEntrezIDs)
# (Time consuming, building the table requires many enrichment tests:)
# dSorensen(pbtGeneLists[[2]], pbtGeneLists[[4]],
#           onto = "CC", GOLevel = 3,
#           geneUniverse = humanEntrezIDs, orgPackg = "org.Hs.eg.db")
# Essentially, the above code makes the same as:
# tab.IRITD3vsKT1 <- buildEnrichTable(pbtGeneLists[[2]], pbtGeneLists[[4]],
#                                     onto = "CC", GOLevel = 3,
#                                geneUniverse = humanEntrezIDs, orgPackg = "org.Hs.eg.db")
# dSorensen(tab.IRITD3vsKT1)
# (Quite time consuming, all pairwise dissimilarities:)
# dSorensen(pbtGeneLists,
#           onto = "CC", GOLevel = 3,
#           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.

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):

n11 n01
n10 n00,

The subindex '11' corresponds to those GO items enriched in both lists, '01' to items enriched in the second list but not in the first one, '10' to items enriched in the first list but not enriched in the second one and '00' corresponds to those GO items 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. 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" 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)

- `table`: S3 method for class "table"
- `matrix`: S3 method for class "matrix"
- `numeric`: S3 method for class "numeric"
- `character`: S3 method for class "character"
- `list`: S3 method for class "list"
- `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 items in ontology BP at level 3.
data(tab_atlas.sanger_BP3)
?tab_atlas.sanger_BP3
duppSorensen(tab_atlas.sanger_BP3)
dSorensen(tab_atlas.sanger_BP3) + qnorm(0.95) * seSorensen(tab_atlas.sanger_BP3)
# Using the bootstrap approximation instead of the normal approximation to
# the sampling distribution of (^d - d) / se(^d):
```

```
duppSorensen(tab_atlas.sanger_BP3, 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(pbtGeneLists)
# ?pbtGeneLists
# data(humanEntrezIDs)
# duppSorensen(pbtGeneLists[[2]], pbtGeneLists[[4]],
#              onto = "CC", GOLevel = 5,
#              geneUniverse = humanEntrezIDs, orgPackg = "org.Hs.eg.db")
# Even more time consuming (all pairwise values):
# duppSorensen(pbtGeneLists,
#              onto = "CC", GOLevel = 5,
#              geneUniverse = humanEntrezIDs, orgPackg = "org.Hs.eg.db")
```

`enrichOnto` *enrichOnto*

Description

This function performs standard tests of enrichment from a gene list

Usage

```
enrichOnto(
  geneList,
  geneUniverse,
  orgPackage = "org.Hs.eg.db",
  onto = c("BP", "MF", "CC"),
  pAdjustMeth = "BH",
  pvalCutoff = 0.01,
  qvalCutoff = 0.05
)
```

Arguments

`geneList`	character vector containing a FIRST gene list of entrez IDs
`geneUniverse`	character vector containing all genes from where geneLists have been extracted
`orgPackage`	A string wih the name of the annotation package
`onto`	string describing the ontology. Belongs to c('BP', 'MF', 'CC', 'ANY')
`pAdjustMeth`	string describing the adjust method. Belongs to c('BH', 'BY', 'Bonf')
`pvalCutoff`	A numeric value
`qvalCutoff`	A numeric value

Value

standard tests of enrichment from a gene list

`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. This checking is performed by means of function nice2x2Table.
y	an object of class "character" representing a list of gene identifiers.

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):

n11 n10
n01 n00,

The subindex '11' corresponds to those GO items enriched in both lists, '01' to items enriched in the second list but not in the first one, '10' to items enriched in the first list but not enriched in the second one and '00' corresponds to those GO items 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. 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" 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. 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)

- `table`: S3 method for class "table"
- `matrix`: S3 method for class "matrix"
- `numeric`: S3 method for class "numeric"
- `character`: S3 method for class "character"
- `list`: S3 method for class "list"
- `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 items in ontology BP at level 3.
data(tab_atlas.sanger_BP3)
tab_atlas.sanger_BP3
equivTestSorensen(tab_atlas.sanger_BP3)
# Bootstrap test:
equivTestSorensen(tab_atlas.sanger_BP3, 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)
# ?pbtGeneLists
# Gene universe:
# data(humanEntrezIDs)
# equivTestSorensen(pbtGeneLists[["IRITD3"]], pbtGeneLists[["IRITD5"]],
#                   geneUniverse = humanEntrezIDs, orgPackg = "org.Hs.eg.db",
#                   onto = "CC", GOLevel = 5)
# Bootstrap instead of normal approximation test:
# equivTestSorensen(pbtGeneLists[["IRITD3"]], pbtGeneLists[["IRITD5"]],
#                   geneUniverse = humanEntrezIDs, orgPackg = "org.Hs.eg.db",
#                   onto = "CC", GOLevel = 5,
#                   boot = TRUE)

# Essentially, the above code makes:
# IRITD3vs5.CC5 <- buildEnrichTable(pbtGeneLists[["IRITD3"]], pbtGeneLists[["IRITD5"]],
#                                   geneUniverse = humanEntrezIDs, orgPackg = "org.Hs.eg.db",
#                                        onto = "CC", GOLevel = 5)
# IRITD3vs5.CC5
# equivTestSorensen(IRITD3vs5.CC5)
# equivTestSorensen(IRITD3vs5.CC5, boot = TRUE)
# (Note that building first the contingency table may be advantageous to save time!)

# All pairwise equivalence tests:
# equivTestSorensen(pbtGeneLists,
#                   geneUniverse = humanEntrezIDs, orgPackg = "org.Hs.eg.db",
#                   onto = "CC", GOLevel = 5)


# 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)
```

`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 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 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)

- `equivSDhtest`: S3 method for class "equivSDhtest"
- `equivSDhtestList`: S3 method for class "equivSDhtestList"
- `AllEquivSDhtest`: S3 method for class "AllEquivSDhtest"

Examples

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

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

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

```
# All Sorensen-Dice dissimilarities:
getDissimilarity(cancerEquivSorensen)
getDissimilarity(cancerEquivSorensen, simplify = FALSE)

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

- `equivSDhtest`: S3 method for class "equivSDhtest"
- `equivSDhtestList`: S3 method for class "equivSDhtestList"
- `AllEquivSDhtest`: S3 method for class "AllEquivSDhtest"

See Also

`getNboot`

Examples

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

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

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

getEffNboot(waldman_atlas.BP.4)
getEffNboot(boot.waldman_atlas.BP.4)
getNboot(boot.waldman_atlas.BP.4)

# All pairwise equivalence tests at level 4 of the BP ontology
data(BP.4)
?BP.4
class(BP.4)
# This may correspond to a call like:
# BP.4 <- equivTestSorensen(allOncoGeneLists,
#                           geneUniverse = humanEntrezIDs, orgPackg = "org.Hs.eg.db",
#                           onto = "BP", GOLevel = 4)
boot.BP.4 <- upgrade(BP.4, boot = TRUE)
getEffNboot(BP.4)
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(cancerEquivSorensen)
# ?cancerEquivSorensen
# class(cancerEquivSorensen)
# This may correspond to code like:
# (By default, the tests are iterated over all GO ontologies and for levels 3 to 10)
# cancerEquivSorensen <- allEquivTestSorensen(allOncoGeneLists,
#                                             geneUniverse = humanEntrezIDs,
#                                             orgPackg = "org.Hs.eg.db",
#                                             boot = TRUE)
# boot.cancerEquivSorensen <- upgrade(cancerEquivSorensen, boot = TRUE)
# Number of effective bootstrap replicates for all tests:
# getEffNboot(boot.cancerEquivSorensen)
# getEffNboot(boot.cancerEquivSorensen, simplify = FALSE)

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

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

- `equivSDhtest`: S3 method for class "equivSDhtest"
- `equivSDhtestList`: S3 method for class "equivSDhtestList"
- `AllEquivSDhtest`: S3 method for class "AllEquivSDhtest"

See Also

`getEffNboot`

Examples

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

# Not a bootstrap test, first upgrade to a bootstrap test:
```

```
boot.waldman_atlas.BP.4 <- upgrade(waldman_atlas.BP.4, boot = TRUE)

getNboot(waldman_atlas.BP.4)
getNboot(boot.waldman_atlas.BP.4)

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

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

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

- `equivSDhtest`: S3 method for class "equivSDhtest"
- `equivSDhtestList`: S3 method for class "equivSDhtestList"
- `AllEquivSDhtest`: S3 method for class "AllEquivSDhtest"

Examples

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

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

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

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

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

- `equivSDhtest`: S3 method for class "equivSDhtest"
- `equivSDhtestList`: S3 method for class "equivSDhtestList"
- `AllEquivSDhtest`: S3 method for class "AllEquivSDhtest"

Examples

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

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

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

```
# All standard errors of the Sorensen-Dice dissimilarity estimates:
getSE(cancerEquivSorensen)
getSE(cancerEquivSorensen, simplify = FALSE)

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

- equivSDhtest: S3 method for class "equivSDhtest"
- equivSDhtestList: S3 method for class "equivSDhtestList"
- AllEquivSDhtest: S3 method for class "AllEquivSDhtest"

Examples

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

# All pairwise equivalence tests at level 4 of the BP ontology
data(BP.4)
?BP.4
class(BP.4)
# This may correspond to a call like:
```

```
# BP.4 <- equivTestSorensen(allOncoGeneLists,
#                           geneUniverse = humanEntrezIDs, orgPackg = "org.Hs.eg.db",
#                           onto = "BP", GOLevel = 4)
getTable(BP.4)

# Equivalence test iterated over all GO ontologies and levels 3 to 10:
data(cancerEquivSorensen)
?cancerEquivSorensen
class(cancerEquivSorensen)
# This may correspond to code like:
# cancerEquivSorensen <- 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(cancerEquivSorensen)
# Contingency tables only for some GO ontologies, levels or pairs of gene lists:
getTable(cancerEquivSorensen, GOLevel = "level 6")
getTable(cancerEquivSorensen, GOLevel = 6)
getTable(cancerEquivSorensen, GOLevel = seq.int(4,6), listNames = c("waldman", "sanger"))
getTable(cancerEquivSorensen, GOLevel = "level 6", onto = "BP")
getTable(cancerEquivSorensen, 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)

- equivSDhtest: S3 method for class "equivSDhtest"
- equivSDhtestList: S3 method for class "equivSDhtestList"
- AllEquivSDhtest: S3 method for class "AllEquivSDhtest"

Examples

```
# Dataset 'allOncoGeneLists' contains the result of the equivalence test between gene lists
# 'waldman' and 'atlas', at level 4 of the BP ontology:
data(waldman_atlas.BP.4)
waldman_atlas.BP.4
class(waldman_atlas.BP.4)
# This may correspond to the result of code like:
# waldman_atlas.BP.4 <- equivTestSorensen(
#   allOncoGeneLists[["waldman"]], allOncoGeneLists[["atlas"]],
```

```
#   geneUniverse = humanEntrezIDs, orgPackg = "org.Hs.eg.db",
#   onto = "BP", GOLevel = 4, listNames = c("waldman", "atlas"))
# (But results may vary according to GO updating)
getUpper(waldman_atlas.BP.4)

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

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

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

GOIDsInLevel *GOIDsInLevel*

Description

This function extends getGOLevel returning only GO identifiers appearing between GO ancestors of at least one GeneList

Usage

```
GOIDsInLevel(
  GOLev,
  onto,
  geneList1 = NULL,
  geneList2 = NULL,
  orgPackage = NULL,
  restricted = TRUE
)
```

Arguments

GOLev	An integer
onto	string describing the ontology. Belongs to c('BP', 'MF', 'CC', 'ANY')
geneList1	character vector containing a FIRST gene list of entrez IDs
geneList2	character vector containing a SECOND gene list of entrez IDs
orgPackage	A string wih the name of the annotation package
restricted	Boolean variable to decide how tabulation is performed.

Value

GO identifiers appearing between GO ancestors of at least one GeneList

---

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

For the moment, the gosorensen package provides the following functions:

buildEnrichTable Build an enrichment contingency table from two gene lists

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

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

`hgu133plus2EntrezIDs` *Entrez Identifiers for all human genes contained in the hgu133plus2.db package.*

Description

Entrez Identifiers for all human genes contained in the hgu133plus2.db package.

Usage

```
data(hgu133plus2EntrezIDs)
```

Format

An object of class `character` of length 22013.

Source

http://bioconductor.org

humanEntrezIDs *Entrez Identifiers for all human genes contained in the org.Hs.eg.db package.*

Description

Entrez Identifiers for all human genes contained in the org.Hs.eg.db package.

Usage

```
data(humanEntrezIDs)
```

Format

An object of class `character` of length 61521.

Source

http://bioconductor.org

kidneyEnrichedGOIDs *GOTerms identifiers obtained from an enrichment analysis performed on kidney Gene lists*

Description

A dataset 5 character vectors representing lists of enriched GO Terms.

Usage

```
data(kidneyEnrichedGOIDs)
```

Format

An object of class `list` of length 5.

Source

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

kidneyGeneLists *Gene lists derived from studies on kidney transplantation rejection*

Description

A dataset 5 character vectors representing 5 gene lists.

Usage

```
data(kidneyGeneLists)
```

Format

An object of class `list` of length 5.

Source

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

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:

n11 n01
n10 n00,

These values are interpreted (always in this order) as n11: number of GO items enriched in both lists, n01: GO items enriched in the second list but not in the first one, n10: items 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 items in two gene lists: "Number of enriched items in list 1 (TRUE, FALSE)" x "Number of enriched items 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)

- `table`: S3 method for class "table"
- `matrix`: S3 method for class "matrix"
- `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)})
```

pbtAllOntosAndLevels	*The Sorensen-Dice test performed on some gene lists possibly related to kidney rejection after transplantation based on non-updated information at* https://rdrr.io/cran/tcgsaseq/man/PBT_gmt.html*, take them just as an illustrative example. Tests performed for all GO ontologies and for GO levels 3 to 10.*

Description

For each ontology and GO level, the result contains the result of all pairwise tests of equivalence between these gene lists.

Usage

```
data(pbtAllOntosAndLevels)
```

Format

An object of class "AllEquivSDhtest" inheriting from class "list". Each one of its elements, named BP, CC and MF respectively, corresponds to a GO ontology. It is itself a list of length 8 whose elements are named as "Level 3" to "Level 10". For each combination of ontology and level, there is an object of class "equivSDhtestList" codifying the result of all pairwise tests between these kidney rejection gene lists.

Examples

```
# This code may help to understand the structure of these data:
data(pbtAllOntosAndLevels)
?pbtAllOntosAndLevels
names(pbtAllOntosAndLevels)
names(pbtAllOntosAndLevels$BP)
names(pbtAllOntosAndLevels$BP$`level 4`)
class(pbtAllOntosAndLevels$BP$`level 4`)
pbtAllOntosAndLevels$BP$`level 4`
names(pbtAllOntosAndLevels$BP$`level 4`$KT1)
class(pbtAllOntosAndLevels$BP$`level 4`$KT1)
class(pbtAllOntosAndLevels$BP$`level 4`$KT1$IRITD5)
pbtAllOntosAndLevels$BP$`level 4`$KT1$IRITD5
```

pbtGeneLists *5 gene lists possibly related with kidney transplant rejection*

Description

An object of class "list" of length 5. 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 gene identifiers.

Source

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

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.

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):

$$\begin{matrix} n11 & n10 \\ n01 & n00, \end{matrix}$$

The subindex '11' corresponds to those GO items enriched in both lists, '01' to items enriched in the second list but not in the first one, '10' to items enriched in the first list but not enriched in the second one and '00' corresponds to those GO items 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. 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" 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)

- `table`: S3 method for class "table"
- `matrix`: S3 method for class "matrix"
- `numeric`: S3 method for class "numeric"
- `character`: S3 method for class "character"
- `list`: S3 method for class "list"
- `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 'Cangenes' dataset. Table of joint enrichment
# of GO items in ontology BP at level 3.
data(tab_atlas.sanger_BP3)
tab_atlas.sanger_BP3
dSorensen(tab_atlas.sanger_BP3)
seSorensen(tab_atlas.sanger_BP3)

# 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)
# ?pbtGeneLists
# Standard error of the sample Sorensen-Dice dissimilarity, directly from
```

```
# two gene lists, from scratch:
# seSorensen(pbtGeneLists[[2]], pbtGeneLists[[4]],
#            onto = "CC", GOLevel = 5,
#            geneUniverse = humanEntrezIDs, orgPackg = "org.Hs.eg.db")
# Essentially, the above code makes the same as:
# tab.IRITD3vsKT1 <- buildEnrichTable(pbtGeneLists[[2]], pbtGeneLists[[4]],
#                                     onto = "CC", GOLevel = 5,
#                                geneUniverse = humanEntrezIDs, orgPackg = "org.Hs.eg.db")
# tab.IRITD3vsKT1
# seSorensen(tab.IRITD3vsKT1)

# All pairwise standard errors (quite time consuming):
# seSorensen(pbtGeneLists,
#            onto = "CC", GOLevel = 5,
#            geneUniverse = humanEntrezIDs, orgPackg = "org.Hs.eg.db")
```

`tab_atlas.sanger_BP3` *Cross-tabulation of enriched GO items at level 3 of ontology BP in two gene lists*

Description

From the "Cancer gene list" of Bushman Lab, a collection of gene lists related with cancer, for gene lists "Atlas" and "Sanger", this dataset is the cross-tabulation of all GO items of ontology BP at level 3 which are: Enriched in both lists, enriched in sanger but not in atlas, non-enriched in sanger but enriched in atlas and non-enriched in both lists. Take it just as an illustrative example, non up-to-date for changes in the gene lists or changes in the GO.

Usage

```
data(tab_atlas.sanger_BP3)
```

Format

An object of class "table" representing a 2x2 contingency table.

Source

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

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)

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

Examples

```
# Result of the equivalence test between gene lists 'waldman' and 'atlas', in dataset
# 'allOncoGeneLists', at level 4 of the BP ontology:
data(waldman_atlas.BP.4)
waldman_atlas.BP.4
class(waldman_atlas.BP.4)
# This may correspond to the result of code like:
```

```
# data(allOncoGeneLists)
# data(humanEntrezIDs)
# waldman_atlas.BP.4 <- equivTestSorensen(
#   allOncoGeneLists[["waldman"]], allOncoGeneLists[["atlas"]],
#   geneUniverse = humanEntrezIDs, orgPackg = "org.Hs.eg.db",
#   onto = "BP", GOLevel = 4, listNames = c("waldman", "atlas"))
upgrade(waldman_atlas.BP.4, d0 = 1/(1 + 10/9)) # d0 = 0.4737
upgrade(waldman_atlas.BP.4, d0 = 1/(1 + 2*1.25)) # d0 = 0.2857
upgrade(waldman_atlas.BP.4, d0 = 1/(1 + 2*1.25), conf.level = 0.99)

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

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

`waldman_atlas.BP.4` *An example of "equivSDhtest" object resulting from a call to function 'equivSorensenTest'*

Description

The Sorensen-Dice equivalence test between the gene lists "waldman" and "atlas" taken from dataset `allOncoGeneLists` which is automatically charged with this package. To perform the test, the information in these gene lists was summarized by means of contingency tables of mutual GO item enrichment, for all GO items at level 4 of the BP ontology. The tests were performed for an equivalence limit d0 = 0.4444 and a confidence level conf.int = 0.95. Based on a non up-to-date version of these gene lists, take just as an illustrative example.

Usage

```
data(waldman_atlas.BP.4)
```

Format

An object of class "equivSDhtest" inheriting from class "list".

Source

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

Index