Package ‘geecc’
March 22, 2017

Type    Package
Title    Gene Set Enrichment Analysis Extended to Contingency Cubes
Version  1.8.0
Date     2016-09-19
Author   Markus Boenn
Maintainer Markus Boenn <markus.boenn@ufz.de>
Description Use log-linear models to perform hypergeometric and chi-squared tests for gene set enrichments for two (based on contingency tables) or three categories (contingency cubes). Categories can be differentially expressed genes, GO terms, sequence length, GC content, chromosomal position, phyla-strata, divergence-strata, ....
License  GPL (>= 2)
Depends  R (>= 3.3.0), methods
Imports  MASS, hypergeo (>= 1.3.0), gplots, Rcpp (>= 0.11.3), graphics, stats, utils
LinkingTo Rcpp
SystemRequirements Rcpp
Suggests  hgu133plus2.db, GO.db, AnnotationDbi
biocViews BiologicalQuestion, GeneSetEnrichment, WorkflowStep, GO, StatisticalMethod, GeneExpression, Transcription, RNASeq, Microarray
NeedsCompilation yes

R topics documented:

<table>
<thead>
<tr>
<th>topic</th>
<th>pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>geecc-package</td>
<td>2</td>
</tr>
<tr>
<td>concub-class</td>
<td>2</td>
</tr>
<tr>
<td>concubfilter-class</td>
<td>4</td>
</tr>
<tr>
<td>filterConCub</td>
<td>6</td>
</tr>
<tr>
<td>GO2list</td>
<td>6</td>
</tr>
<tr>
<td>marioni</td>
<td>8</td>
</tr>
<tr>
<td>plotConCub</td>
<td>8</td>
</tr>
<tr>
<td>pval2star</td>
<td>10</td>
</tr>
<tr>
<td>runConCub</td>
<td>10</td>
</tr>
<tr>
<td>sortAscii</td>
<td>12</td>
</tr>
</tbody>
</table>

Index 13
Description

This package performs gene set enrichment analyses considering two or three categories. Categories might be regulated genes, sequence length, GC content, GO terms, KEGG pathways and so on.

Author(s)

Markus Boenn Maintainer: Markus Boenn <markus.boenn@ufz.de>

Examples

```r
## a completely artificial example run
## through the routines of the package
##
R <- 500
generate R random gene-ids
ID <- sapply(1:R, function(r){paste( sample(LETTERS, 10), collapse="" ) })
ID <- unique(ID)

#assign artificial differentially expressed genes randomly
category1 <- list( deg.smallFC=sample(ID, 100, rep=FALSE),
deg.hughFC=sample(ID, 100, rep=FALSE) )
#assign artificial GO terms of genes randomly
category2 <- list( go1=sample(ID, 50, replace=FALSE),
go2=sample(ID, 166, replace=FALSE),
go3=sample(ID, 74, replace=FALSE),
go4=sample(ID, 68, replace=FALSE) )
#assign artificial sequence length of genes randomly
LEN <- setNames(sample(seq(100, 1000, 100), length(ID), replace=TRUE), ID)
category3 <- split( ID, f=Factor(LEN, levels=seq(100, 1000, 100)) )
CatList <- list(deg=category1, go=category2, len=category3)

ConCubFilter.obj <- new("concubfilter", names=names(CatList))
ConCub.obj <- new("concub", categories=CatList)
ConCub.obj.2 <- runConCub( obj=ConCub.obj, filter=ConCubFilter.obj, nthreads=1 )
ConCub.obj.3 <- filterConCub( obj=ConCub.obj.2, filter=ConCubFilter.obj )
plotConCub( obj=ConCub.obj.3, filter=ConCubFilter.obj )
x <- getTable(ConCub.obj.3)
```
concub-class

Details

Specifying the background population is crucial for the tests for association between factors. Usually the population is the set of all probe sets represented on a micro-array or the set of all genes in a genome. If an expression set is passed with the population-parameter, all probe sets beginning with the pattern “AFFX” (Affymetrix quality control) are removed.

Objects from the Class

Objects can be created by calls of the form new("concub", ...).

Slots

categories: A named list of named lists. Each item of the outer list represents the two or three categories. Each item of the inner lists represents a variable of the category.

population: A character vector containing the background population. As an alternative, an object with class 'eSet', 'ExpressionSet', or 'DGEList'; background population is then set to the outcome of rownames(population).

keep.empty.vars: A boolean list with names being names of categories.

options: Additional options for individual categories. See Details.

approx: specifies the minimum expected value when an exact hypergeometric test (below) or the chi-squared approximation should be used. Defaults to 0.

null.model: A formula specifying the null-model of the test.

test.result: A list to store test results. Filled up after runConCub.

test.result.filter: A list to store filtered test results. Filled up after filterConCub.

test.result.filter.heatmap: A list to store heatmaps for further manipulation. Filled up after plotConCub.

The last three slots are not set by the user.

Methods

getTable signature(object = "concub"): creates a table containing the results of the tests.

Usage: getTable(object, na.rm=TRUE, dontshow=list())

Arguments:

doject an object of type concub

na.rm logical. If TRUE (the default), rows with NA P-values (or odds ratios) are removed

The resulting table is a data frame with 8 or 10 columns, depending on if a two- or three-way test was applied

1. ‘factor1’: this column has the name of the first category
2. ‘factor2’: this column has the name of the second category
3. ‘factor3’: this column has the name of the third category
4. n.'factor1': number of items in variable of first category
5. n.'factor2': number of items in variable of second category
6. n.'factor3': number of items in variable of third category
7. p.value:(probably adjusted) p-value
8. log2.odds.ratio: log2 of the sample odds ratio
9. n.tags: number of items at position x_{1,1,1} or x_{1,1}
10. tags: items at position x_{1,1,1} or x_{1,1} (e.g. gene identifiers)
You have to run `filterConCub()` before you can get the table. If `filterConCub()` was not run, a warning is shown and `getTable` returns `NULL`.

```r
show signature(object = "concub")
```

**See Also**

GOStats to perform a simple two-way enrichment analysis

**Examples**

```r
showClass("concub")
```

---

**concubfilter-class  Class "concubfilter"**

**Description**

An object of type concubfilter

**Details**

The large number of different filter options (and corresponding getter and setter accessors) makes it necessary to maintain them in a special class. This differs from other packages like GOstats, where arguments for controlling the program and the results are stored in the same object.

**Objects from the Class**

Objects can be created by calls of the form `new("concubfilter", ...)`.  

**Slots**

For a more detailed description of some slots see below.

- `nfact`: a numeric giving the number of factors
- `names`: a character vector giving the name of each factor
- `p.value`: a numeric giving the P-value to be considered. Defaults to 0.1.
- `test.direction`: a character giving the direction of association. Defaults to “two.sided”.
- `minimum.l2or`: a numeric giving the minimum absolute log2 odds ratio to be considered. Defaults to 0.
- `skip.min.group`: a numeric giving the minimum number of tag a group is allowed to have. Defaults to 2.
- `skip.min.obs`: a numeric giving the minimum number at the position of interest allowed. Defaults to 2.
- `skip.zeroobs`: a logical. Defaults to `TRUE`.
- `drop.insignif.layer`: A vector of logicals. By default, all positions are set to `FALSE`.
- `drop.wrongdir.layer`: A vector of logicals. By default, all positions are set to `FALSE`.
- `drop.lowl2or.layer`: A vector of logicals. By default, all positions are set to `FALSE`. 

Methods and slot accessors

Several methods are implemented for class concubfilter. They can be roughly grouped into informative, basic, skip-test, and data-reduction methods.

Individual options can be accessed by the corresponding getter and setter methods, for instance

skip.zeroobs signature(x = "concubfilter"): get current setting to skip test in case of zero cell

skip.zeroobs<- signature(x = "concubfilter"): set a new value to skip test in case of zero cell

Informative methods: Currently only a single method is implemented.

show signature(object = "concubfilter"): a short summary about current filter settings

Basic filters/thresholds:

p.value The maximum P-value that should be taken into account
minimum.l2or The minimum absolute of log2 odds ratio that should be taken into account
test.direction The direction of the association. Can be “two.sided”, “greater” (test for over-representation), or “less” (test for under-representation)

Skip test: The following filters cause a skip of a test, i.e. the test is never run if at least one of the conditions is fullfilled.

skip.zeroobs skip the test, if the position of interest (x_{00} or x_{000}) is zero, i.e. no tag from the population matches the conditions defined at the marginals.
skip.mingroup skip the test, if the groups considered at the position of interest are too small at all.
skip.minobs skip the test, if the position of interest contains less than ‘this value’ entries.

Data reduction: The following filters reduce the amount of outcome of the tests. They are applied to both, the (2 or 3 dimensional) table containing the odds ratios and the table containing the corresponding P-values.

drop.insignif.layer drop all layers in the tables where all P-values are greater than the value defined in ‘p.value’.
drop.wrongdir.layer drop all layers in the tables where all odds ratios are showing into the opposite direction as defined in ‘test.direction’.
drop.lowl2or.layer drop all layers in the tables where all absolutes of the log2 odds ratios are smaller than ‘min.l2or’.

Examples

showClass("concub")
**filterConCub**  
*Filter results from two- or three-way tests*

**Description**
Performs filtering on results from two- or three-way tests

**Usage**
```r
filterConCub(obj, filter, p.adjust.method = "none", verbose=1)
```

**Arguments**
- `obj`: object of type `concub`
- `filter`: object of type `concubfilter`
- `verbose`: An integer specifying the level of verbosity.

**Details**
You have to execute `runConCub` before filtering.

**Value**
an (extended) object of type `concub` with filtered test results

**Examples**
```r
# a character vector listing possible adjustment approaches
p.adjust.methods
```

---

**GO2list**  
*Filter GO and KEGG database*

**Description**
Filter GO and KEGG database and transform database to list

**Usage**
```r
GO2list(dbase, go.cat = NULL, rm = NULL, keep = NULL)
KEGG2list(dbase, rm = NULL, keep = NULL)
GO2offspring(x)
GO2level(x, go.level=-1, relation=c("is_a"))
```
**Arguments**

- **dbase**: A datastructure storing identifiers of GO/KEGG terms and assigned genes. Can be one of a **database** usually of class 'ProbeGo3AnnDbBimap' (as defined in package "AnnotationDbi") or a **named list** with keys being the identifiers and values being genes. A **dataframe** with first column being the identifiers and second column being genes. Additional columns are ignored.
- **x**: a list with keys being the identifiers and values being genes (e.g. output of `GO2list`)
- **go.cat**: GO category ("MF", "BP", "CC") that should be returned and filtered
- **go.level**: Level in the DAG of GO terms. Defaults to "-1" for pass through without modification. Otherwise: a positive integer giving the level at which GO terms should be grouped together.
- **rm**: remove these terms
- **keep**: keep only these terms
- **relation**: relationships in GO hierarchy that should be considered. Defaults to "is_a"

**Details**

The settings for "rm" and "keep" can be combined, allowing for efficient reduction of the number of GO terms and KEGG pathways, respectively.

Providing a named list instead of a database can be useful for non-model organisms, where only a draft Blast2GO-annotation is available. In this case, the names of the list are the GO terms (or KEGG pathways) and the content of each list item is a character vector with tag-ids.

The function `GO2offspring` does the same as the `databaseGO2ALLPROBES` function does (e.g. `hgu133plus2GO2ALLPROBES`). I.e. instead of representing only features (probe sets, genes, ...) assigned to the GO terms directly, it also contains all features assigned to all children (offsprings).

The function `GO2level` groups GO terms together at a more general level to simplify data interpretation and speed up runtime. This function works according to the level option provided by DAVID, but the number of levels is not restricted.

**Value**

A named list with each slot containing the ids for the term or pathway.

**Examples**

```r
library(hgu133plus2.db)
x <- GO2list(dbase=hgu133plus2GO2PROBE, go.cat="CC", rm=c("GO:0000139", "GO:0000790", "GO:0005730", "GO:0005739"))
```
marioni

Affymetrix microarray gene expression data

Description
The experiment aims to detect differentially expressed genes in Affymetrix micro-arrays and RNA-seq data in a comparative study. For this, samples from two tissues (liver and kidney) were compared.

Usage
marioni

Format
A data.frame containing gene expression values from an Affymetrix microarray, including P-values, log2-fold changes and alternative annotations

Value
A data.frame.

Source
http://giladlab.uchicago.edu/data.html

References

Examples
data(marioni)
head(marioni[, 1:5])

plotConCub

Generate a heatmap showing log_2 odds ratios and P-values.

Description
The function generates a heatmap by calling the heatmap.2-function from the gplots-package. Each cell shows the log_2 odds ratio of the test for the corresponding variables. In addition, stars indicate the P-value for this test.

Usage
plotConCub(obj, filter, fix.cat = 1, show=list(), dontshow=list(),
args_heatmap.2 = list(), col = list(range = NULL),
alt.names = list(), t = FALSE)
**plotConCub**

Arguments

- `obj`  
  An object with class concub

- `filter`  
  An object with class concubfilter

- `fix.cat`  
  The heatmap can only visualize a two-dimensional table. In case of three-dimensions, one dimension (category) must be fixed.

- `show`  
  A named list. The names are the names of the categories. Each item is a character vector of variables that should be shown in the plot.

- `dontshow`  
  A named list. The names are the names of the categories. Each item is a character vector of variables that should not be shown in the plot.

- `args_heatmap.2`  
  Arguments passed to `heatmap.2`. Can be used to change size of fonts etc.

- `col`  
  A vector of colors, for instance from `heat.colors`

- `alt.names`  
  Substitute variables by alternative terms. For instance, if variables are artificial ids, they can be substituted by descriptive text for the heatmap.

- `t`  
  logical; transpose matrix for heatmap. Default FALSE.

Value

An object with class concub.

Examples

```r
## a completely artificial example run
## through the routines of the package
##
R <- 500
#generate R random gene-ids
ID <- sapply(1:R, function(r){paste( sample(LETTERS, 10), collapse=" ") })
ID <- unique(ID)
#assign artificial differentially expressed genes randomly
category1 <- list( deg.smallFC=sample(ID, 100, rep=FALSE),
deg.hughFC=sample(ID, 100, rep=FALSE) )
#assign artificial GO terms of genes randomly
category2 <- list( go1=sample(ID, 50, replace=FALSE),
go2=sample(ID, 150, replace=FALSE),
go3=sample(ID, 74, replace=FALSE),
go4=sample(ID, 68, replace=FALSE) )
#assign artificial sequence length of genes randomly
LEN <- setNames(sample(seq(100, 1000, 100), length(ID), replace=TRUE), ID)
category3 <- split( ID, f=factor(LEN, levels=seq(100, 1000, 100)) )
CatList <- list(deg=category1, go=category2, len=category3)
ConCubFilter.obj <- new("concubfilter", names=names(CatList))
ConCub.obj <- new("concub", categories=CatList)
ConCub.obj.2 <- runConCub( obj=ConCub.obj, filter=ConCubFilter.obj, nthreads=1 )
ConCub.obj.3 <- filterConCub( obj=ConCub.obj.2, filter=ConCubFilter.obj )
plotConCub( obj=ConCub.obj.3, filter=ConCubFilter.obj )
```
pval2star  

Transform P-values to stars

Description
Transform P-values to stars

Usage
pval2star(x)

Arguments
x
A matrix of P-values

Details
Use stars as simplification of P-values

Value
A character matrix of same dimension and names as x with stars instead of P-values.

Examples
x <- matrix( runif(25), nrow=5, dimnames=list(LETTERS[1:5], letters[1:5]) )
pval2star(x)

runConCub  

Enrichment analysis on two- or three-way contingency tables.

Description
Perform the enrichment analysis on two- or three-way contingency tables.

Usage
runConCub(obj, filter, nthreads = 2, subset = NULL, verbose=list(output.step=0, show.cat1=FALSE, show.cat2=FALSE, show.cat3=FALSE))

Arguments
obj
an object with class concub
filter
an object with class concubfilter
nthreads
number of threads to use in hypergeom.test
subset
a named list. Restrict enrichment analysis to these category variables
verbose
A list to control verbosity:
runConCub

**output.step:** after how many variables passed of category 2 a control output should be printed

**show.cat1:** show current level of category 1

**show.cat2:** show current level of category 2

**show.cat3:** show current level of category 3

**Details**

This function applies a test for association for all combinations of all variables of all categories to be tested. Depending on the settings in the concubfilter-object, a one-sided or two-sided test is made, using the exact hypergeometric test as implemented in the hypergea-package if the smallest expected value is smaller than 5, or using the chi-squared test as implemented in the loglm-function implemented in the MASS-package. The minimum expected value can be changed in the concub-object by the user (approx-parameter). In this function only filter-settings those filter settings are used, which skip the tests.

**Value**

An object with class concub.

**Examples**

```r
##
## a completely artificial example run
## through the routines of the package
##
R <- 500
#generate R random gene-ids
ID <- sapply(1:R, function(r){paste( sample(LETTERS, 10), collapse="" ) } )
ID <- unique(ID)

#assign artificial differentially expressed genes randomly
category1 <- list( deg.smallFC=sample(ID, 100, rep=FALSE),
deq.hughFC=sample(ID, 100, rep=FALSE) )
#assign artificial GO terms of genes randomly
category2 <- list( go1=sample(ID, 50, replace=FALSE),
go2=sample(ID, 166, replace=FALSE),
go3=sample(ID, 74, replace=FALSE),
go4=sample(ID, 68, replace=FALSE) )
#assign artificial sequence length of genes randomly
LEN <- setNames(sample(seq(100, 1000, 100), length(ID), replace=TRUE), ID)
category3 <- split( ID, f=factor(LEN, levels=seq(100, 1000, 100)) )
CatList <- list(deg=category1, go=category2, len=category3)

ConCubFilter.obj <- new("concubfilter", names=names(CatList))
ConCub.obj <- new("concub", categories=CatList)
ConCub.obj.2 <- runConCub( obj=ConCub.obj, filter=ConCubFilter.obj, nthreads=1 )
ConCub.obj.2
```
sortAscii

Optimized operations of sets of character-vectors

Description

Sort and use pre-sorted character vectors in set-operations

Usage

sortAscii(x)
intersectPresort(pop, x)
setdiffPresort(pop, x)

Arguments

x               an unsorted vectors of strings
pop             a sorted vector of strings

Details

By default, sorting is done lexicographically in R. The routine sortAscii does sorting according to the ASCII-order as done in C/C++.

For routines intersectPresort and setdiffPresort the first argument has to be sorted according to ASCII-order. This first argument is expected to be large compared to the second argument. Both functions are wrappers for optimized C++-functions performing the set-operation.

Value

A character-vector. In case of intersectPresort and setdiffPresort, these vectors are un-named.

Examples

AA <- matrix( sample( c(LETTERS, letters), 10*30000, rep=TRUE ), ncol=10 )
A <- unique(apply(AA, 1, paste, collapse=""))
B <- sample(AA, 100, replace=FALSE); B <- c(B, "1234")

res <- intersectPresort( sortAscii(A), B )
Index

*Topic classes
  concub-class, 2
  concubfilter-class, 4
*Topic datasets
  marioni, 8
*Topic package
  geecc-package, 2
  concub-class, 2
  concubfilter-class, 4

drop.insignif.layer
  (concubfilter-class), 4
drop.insignif.layer, concubfilter-method
  (concubfilter-class), 4
drop.insignif.layer<-
  (concubfilter-class), 4
drop.insignif.layer<-, concubfilter-method
  (concubfilter-class), 4
drop.lowl2or.layer
  (concubfilter-class), 4
drop.lowl2or.layer, concubfilter-method
  (concubfilter-class), 4
drop.lowl2or.layer<-
  (concubfilter-class), 4
drop.lowl2or.layer<-, concubfilter-method
  (concubfilter-class), 4
drop.wrongdir.layer
  (concubfilter-class), 4
drop.wrongdir.layer, concubfilter-method
  (concubfilter-class), 4
drop.wrongdir.layer<-
  (concubfilter-class), 4
drop.wrongdir.layer<-, concubfilter-method
  (concubfilter-class), 4

filterConCub, 6
geecc (geecc-package), 2
geecct-package, 2
getTable (concub-class), 2
getTable, concub-method (concub-class), 2
GO2level (GO2list), 6
GO2list, 6

G02offspring (GO2list), 6
heat.colors, 9
heatmap, 2, 9
initialize, concub-method
  (concub-class), 2
initialize, concubfilter-method
  (concubfilter-class), 4
intersectPresort (sortAscii), 12
KEGG2list (GO2list), 6
marioni, 8
minimum.l2or (concubfilter-class), 4
minimum.l2or, concubfilter-method
  (concubfilter-class), 4
minimum.l2or<-
  (concubfilter-class), 4
minimum.l2or<-, concubfilter-method
  (concubfilter-class), 4
p.adjust.methods, 6
p.value (concubfilter-class), 4
p.value, concubfilter-method
  (concubfilter-class), 4
p.value<-
  (concubfilter-class), 4
p.value<-, concubfilter-method
  (concubfilter-class), 4
plotConCub, 8
pval2star, 10
runConCub, 6, 10
setdiffPresort (sortAscii), 12
show, concub-method (concub-class), 2
show, concubfilter-method
  (concubfilter-class), 4
skip.min.group (concubfilter-class), 4
skip.min.group, concubfilter-method
  (concubfilter-class), 4
skip.min.group<-
  (concubfilter-class), 4
skip.min.group<-, concubfilter-method
  (concubfilter-class), 4
skip.min.obs (concubfilter-class), 4
INDEX

skip.min.obs, concubfilter-method
  (concubfilter-class), 4
skip.min.obs<-(concubfilter-class), 4
skip.min.obs<-, concubfilter-method
  (concubfilter-class), 4
skip.zeroobs (concubfilter-class), 4
skip.zeroobs, concubfilter-method
  (concubfilter-class), 4
skip.zeroobs<-(concubfilter-class), 4
skip.zeroobs<-, concubfilter-method
  (concubfilter-class), 4
sortAscii, 12

test.direction (concubfilter-class), 4
test.direction, concubfilter-method
  (concubfilter-class), 4
test.direction<-(concubfilter-class), 4
test.direction<-, concubfilter-method
  (concubfilter-class), 4