Package ‘simplifyEnrichment’

May 29, 2024

**Type**  Package

**Title**  Simplify Functional Enrichment Results

**Version**  1.14.0

**Date**  2023-11-06

**Depends**  R (>= 3.6.0), BiocGenerics, grid

**Imports**  GOSemSim, ComplexHeatmap (>= 2.7.4), circlize, GetoptLong, digest, tm, GO.db, org.Hs.eg.db, AnnotationDbi, slam, methods, clue, grDevices, graphics, stats, utils, proxyC, Matrix, cluster (>= 1.14.2), colorspace, GlobalOptions (>= 0.1.0)

**Suggests**  knitr, ggplot2, cowplot, mclust, apcluster, MCL, dbscan, igraph, gridExtra, dynamicTreeCut, testthat, gridGraphics, clusterProfiler, mSigDB, DOSE, DO.db, reactome.db, flexclust, BiocManager, InteractiveComplexHeatmap (>= 0.99.11), shiny, shinydashboard, cola, hu6800.db, rmarkdown, genefilter, gridtext, fpc

**Description**  A new clustering algorithm, `binary cut`, for clustering similarity matrices of functional terms is implemented in this package. It also provides functions for visualizing, summarizing and comparing the clusterings.

**biocViews**  Software, Visualization, GO, Clustering, GeneSetEnrichment


**VignetteBuilder**  knitr

**License**  MIT + file LICENSE

**git_url**  https://git.bioconductor.org/packages/simplifyEnrichment

**git_branch**  RELEASE_3_19

**git_last_commit**  019c312

**git_last_commit_date**  2024-04-30

**Repository**  Bioconductor 3.19

**Date/Publication**  2024-05-28
Author  Zuguang Gu [aut, cre] (<https://orcid.org/0000-0002-7395-8709>)
Maintainer  Zuguang Gu <z.gu@dkfz.de>

Contents

all_clustering_methods  .............................................. 3
anno_word_cloud  .................................................. 4
anno_word_cloud_from_GO  .......................................... 5
area_above_ecdf  ................................................... 6
binary_cut  .......................................................... 7
cluster_by_apcluster  .............................................. 8
cluster_by_dynamicTreeCut  ......................................... 8
cluster_by_hdbscan  ............................................... 9
cluster_by_igraph  ............................................... 10
cluster_by_kmeans  ............................................... 11
cluster_by_MCL  .................................................. 11
cluster_by_mclust  ................................................ 12
cluster_by_pam  ................................................... 13
cluster_terms  ..................................................... 13
cmp_make_clusters  ................................................. 15
cmp_make_plot  ..................................................... 16
cmp_make_clusters  ................................................. 17
cmp_make_plot  ..................................................... 18
diff_node_apply  .................................................... 19
difference_score  ................................................... 20
DO_similarity  ....................................................... 21
diff_node_apply  .................................................... 22
difference_score  ................................................... 23
export_to_shiny_app  ................................................. 23
GO_similarity  ....................................................... 23
guess_ont  .......................................................... 24
ht_clusters  ........................................................ 25
ht_clusters  ........................................................ 26
keyword_enrichment_from_GO  ..................................... 27
partition_by_hclust  ............................................... 28
partition_by_kmeans  ............................................... 29
partition_by_kmeanspp  .......................................... 30
partition_by_pam  .................................................. 30
plot_binary_cut  .................................................... 31
random_DO  .......................................................... 32
random_GO  .......................................................... 32
register_clustering_methods  ..................................... 33
remove_clustering_methods  ....................................... 34
reset_clustering_methods  ......................................... 34
scale_fontsize  ..................................................... 35
select_cutoff  ....................................................... 36
se_opt  .............................................................. 37
simplifyEnrichment  ................................................ 37
Description

All clustering methods

Usage

all_clustering_methods()

Details

The default clustering methods are:

- kmeans see cluster_by_kmeans.
- dynamicTreeCut see cluster_by_dynamicTreeCut.
- mclust see cluster_by_mclust.
- apcluster see cluster_by_apcluster.
- hdbscan see cluster_by_hdbscan.
- fast_greedy see cluster_by_igraph.
- louvain see cluster_by_igraph.
- walktrap see cluster_by_igraph.
- MCL see cluster_by_MCL.
- binary_cut see binary_cut.

Value

A vector of method names.
See Also

New methods can be added by `register_clustering_methods`.

Examples

```r
all_clustering_methods()
```

---

**anno_word_cloud**  
Word cloud annotations

### Description

Word cloud annotations

### Usage

```r
anno_word_cloud(align_to, term, exclude_words = NULL, max_words = 10,  
               word_cloud_grob_param = list(), fontsize_range = c(4, 16), value_range = NULL,  
               bg_gp = gpar(fill = "#DDDDDD", col = "#AAAAAA"), side = c("right", "left"),  
               add_new_line = FALSE, count_words_param = list(), ..., return_gbl = FALSE)
```

### Arguments

- **align_to**: How to align the annotations to the heatmap. Similar as in `anno_link`, the value of `align_to` can be a list of row indices or a categorical vector where each vector in the list corresponds to a word cloud. If it is a categorical vector, rows with the same level correspond to a same word cloud. If `align_to` is a categorical vector and `term` is a list, names of `term` should have overlap to the levels in `align_to`. When `align_to` is set as a categorical vector, normally the same value is set to `row_split` in the main heatmap so that each row slice can correspond to a word cloud.

- **term**: The description text used for constructing the word clouds. The value should have the same format as `align_to`. If `align_to` is a list, `term` should also be a list. In this case, the length of vectors in `term` is not necessarily the same as in `align_to`. E.g. `length(term[[1]])` is not necessarily equal to `length(align_to[[1]])`. If `align_to` is a categorical vector, `term` should also be a character vector with the same length as `align_to`. To make it more general, when `align_to` is a list, `term` can also be a list of data frames where the first column contains keywords and the second column contains numeric values that will be mapped to font sizes in the word clouds.

- **exclude_words**: The words excluded for constructing word cloud.

- **max_words**: Maximal number of words visualized in the word cloud.

- **word_cloud_grob_param**: A list of graphics parameters passed to `word_cloud_grob`.

- **fontsize_range**: The range of the font size. The value should be a numeric vector with length two. The font size interpolation is linear.
## Word cloud annotations from GO

### Description

Word cloud annotations from GO

### Value Range

The range of values to map to font sizes.

### BG GP

Graphics parameters for controlling the background.

### Side

Side of the annotation relative to the heatmap.

### Add New Line

Whether to add new line after every word? If TRUE, each word will be in a separated line.

### Count Words Param

A list of parameters passed to `count_words`.

### Details

The word cloud annotation is constructed by `anno_link`.

If the annotation is failed to construct or no keyword is found, the function returns a `anno_empty` with 1px width.

English stop words, punctuation and numbers are removed by default when counting words. As specific stop words might coincide with gene or pathway names, and numbers in genes names might be meaningful it is recommended to adjust this behaviour by passing appropriate arguments to the `count_words` function using `count_words_param`.

### Examples

```r
gm = readRDS(system.file("extdata", "random_GO_BP_sim_mat.rds", package = "simplifyEnrichment"))
go_id = rownames(gm)
go_term = AnnotationDbi::select(GO.db::GO.db, keys = go_id, columns = "TERM")$TERM

split = sample(letters[1:4], 100, replace = TRUE)
align_to = split(1:100, split)
term = lapply(letters[1:4], function(x) sample(go_term, sample(100:400, 1)))
names(term) = letters[1:4]

require(ComplexHeatmap)
mat = matrix(rnorm(100*10), nrow = 100)
Heatmap(mat, cluster_rows = FALSE, row_split = split,
right_annotation = rowAnnotation(foo = anno_word_cloud(align_to, term)))
```
Usage

anno_word_cloud_from_GO(align_to, go_id, stat = c("pvalue", "count"),
min_stat = ifelse(stat == "count", 5, 0.05),
term = NULL, exclude_words = NULL, ...)

Arguments

align_to
The same format as in anno_word_cloud.

go_id
The value should be in the same format as align_to. If go_id is a vector, it should have the same length as align_to, and if go_id is a list, note, e.g. length(go_id[[1]]) is not necessarily equal to length(align_to[[1]]). If align_to is a categorical vector and go_id is a list, names of go_id should have overlap to the levels in align_to.

min_stat
Minimal value for stat for selecting keywords.

stat
What type of value to map to font sizes of the keywords. There are two possible values, "pvalue": enrichment is applied to keywords and -log10(p-value) is used to map to font size; "count": simply word frequency of keywords.

term
Alternatively the GO description can be set via the term argument. The same format as in anno_word_cloud.

exclude_words
The words exluded for constructing word cloud. Some words are internally excluded: c("via", "protein", "factor", "side", "type", "specific").

...
All other arguments passed to anno_word_cloud.

Examples

# There is no example
NULL

area_above_ecdf

Area above the eCDF curve

Description

Area above the eCDF curve

Usage

area_above_ecdf(x)

Arguments

x
A vector of similarity values.
**Details**

Denote \( F(x) \) as the eCDF (empirical Cumulative Distribution Function) of the similarity vector \( x \), this function calculates the area above the eCDF curve, which is \( 1 - \int_0^1 F(x)dx \).

**Value**

A numeric value.

**Examples**

```r
# There is no example
NULL
```

---

**binary_cut**  
*Cluster functional terms by recursively binary cutting the similarity matrix*

**Description**

Cluster functional terms by recursively binary cutting the similarity matrix

**Usage**

```r
binary_cut(mat, value_fun = area_above_ecdf, partition_fun = partition_by_pam,  
cutoff = 0.85, try_all_partition_fun = FALSE, partial = FALSE)
```

**Arguments**

- `mat`  
  A similarity matrix.
- `value_fun`  
  A function that calculates the scores for the four submatrices on a node.
- `partition_fun`  
  A function to split each node into two groups. Pre-defined functions in this package are `partition_by_kmeanspp`, `partition_by_pam` and `partition_by_hclust`.
- `cutoff`  
  The cutoff for splitting the dendrogram.
- `try_all_partition_fun`  
  Different `partition_fun` gives different clusterings. If the value of `try_all_partition_fun` is set to `TRUE`, the similarity matrix is clustered by three partitioning method: `partition_by_pam`, `partition_by_kmeanspp` and `partition_by_hclust`. The clustering with the highest difference score is finally selected as the final clustering.
- `partial`  
  Whether to generate the complete clustering or the clustering stops when submatrices cannot be split anymore.

**Value**

A vector of cluster labels (in numeric).
Examples

```
mat = readRDS(system.file("extdata", "random_GO_BP_sim_mat.rds",
    package = "simplifyEnrichment"))
binary_cut(mat)
```

---

**cluster_by_apcluster**  
*Cluster similarity matrix by apcluster*

**Description**

Cluster similarity matrix by apcluster

**Usage**

```
cluster_by_apcluster(mat, s = apcluster::negDistMat(r = 2), ...)
```

**Arguments**

- **mat**  
The similarity matrix.
- **s**  
Passed to the s argument in **apcluster**.
- **...**  
Other arguments passed to **apcluster**.

**Value**

A vector of cluster labels (in numeric).

**Examples**

```
# There is no example
NULL
```

---

**cluster_by_dynamicTreeCut**  
*Cluster similarity matrix by dynamicTreeCut*

**Description**

Cluster similarity matrix by dynamicTreeCut

**Usage**

```
cluster_by_dynamicTreeCut(mat, minClusterSize = 5, ...)
```
cluster_by_hdbscan

Arguments

mat      The similarity matrix.
minClusterSize  Minimal number of objects in a cluster. Pass to `cutreeDynamic`.
...      Other arguments passed to `cutreeDynamic`.

Value

A vector of cluster labels (in numeric).

Examples

# There is no example
NULL

cluster_by_hdbscan  Cluster similarity matrix by hdbscan

Description

Cluster similarity matrix by hdbscan

Usage

cluster_by_hdbscan(mat, minPts = 5, ...)

Arguments

mat      The similarity matrix.
minPts  Passed to the minPts argument in `hdbscan`.
...      Other arguments passed to `hdbscan`.

Value

A vector of cluster labels (in numeric).

Examples

# There is no example
NULL
cluster_by_igraph

Cluster similarity matrix by graph community detection methods

Description

Cluster similarity matrix by graph community detection methods

Usage

cluster_by_igraph(mat,  
    method = c("fast_greedy",  
              "leading_eigen",  
              "louvain",  
              "walktrap"),  
    ...)  

Arguments

mat The similarity matrix.  
method The community detection method.  
... Other arguments passed to the corresponding community detection function, see Details.

Details

The symmetric similarity matrix can be treated as an adjacency matrix and constructed as a graph/network with the similarity values as the weight of the edges. Thus, clustering the similarity matrix can be treated as detecting clusters/modules/communities from the graph.  
Four methods implemented in igraph package can be used here:

fast_greedy uses cluster_fast_greedy.  
leading_eigen uses cluster_leading_eigen.  
louvain uses cluster_louvain.  
walktrap uses cluster_walktrap.

Value

A vector of cluster labels (in numeric).

Examples

# There is no example
NULL
Description

Cluster similarity matrix by k-means clustering

Usage

cluster_by_kmeans(mat, max_k = max(2, min(round(nrow(mat)/5), 100)), ...)

Arguments

mat The similarity matrix.
max_k maximal k for k-means clustering.
... Other arguments passed to kmeans.

Details

The best number of k for k-means clustering is identified according to the "elbow" or "knee" method on the distribution of within-cluster sum of squares (WSS) at each k.

Value

A vector of cluster labels (in numeric).

Examples

# There is no example
NULL

Description

Cluster similarity matrix by MCL

Usage

cluster_by_MCL(mat, addLoops = TRUE, ...)

cluster_by_MCL Cluster similarity matrix by MCL

Description

Cluster similarity matrix by MCL

Usage

cluster_by_MCL(mat, addLoops = TRUE, ...)
cluster_by_mclust

Arguments

mat The similarity matrix.
addLoops Passed to the addLoops argument in mcl.
... Other arguments passed to mcl.

Value

A vector of cluster labels (in numeric).

Examples

# There is no example
NULL

cluster_by_mclust Cluster similarity matrix by mclust

Description

Cluster similarity matrix by mclust

Usage

cluster_by_mclust(mat, G = seq_len(max(2, min(round(nrow(mat)/5), 100))), ...)

Arguments

mat The similarity matrix.
G Passed to the G argument in Mclust.
... Other arguments passed to Mclust.

Value

A vector of cluster labels (in numeric).

Examples

# There is no example
NULL
# cluster_by_pam

Cluster similarity matrix by pam clustering

## Usage

```r
cluster_by_pam(mat, max_k = max(2, min(round(nrow(mat)/10), 100)), ...)
```

## Arguments

- `mat` The similarity matrix.
- `max_k` maximal k for pam clustering.
- `...` Other arguments passed to `pamk`.

## Details

PAM is applied by `pamk` which can automatically select the best k.

## Value

A vector of cluster labels (in numeric).

## Examples

```r
# There is no example
NULL
```

---

# cluster_terms

Cluster functional terms

## Usage

```r
cluster_terms(mat, method = "binary_cut", control = list(), catch_error = FALSE, verbose = TRUE)
```
Arguments

mat  A similarity matrix.
method Method for clustering the matrix.
control A list of parameters passed to the corresponding clustering function.
catch_error Internally used.
verbose Whether to print messages.

Details

The following methods are the default:

kmeans see cluster_by_kmeans.
pam see cluster_by_pam.
dynamicTreeCut see cluster_by_dynamicTreeCut.
mclust see cluster_by_mclust.
apcluster see cluster_by_apcluster.
hdbscan see cluster_by_hdbscan.
leading_eigen see cluster_by_igraph.
louvain see cluster_by_igraph.
walktrap see cluster_by_igraph.
MCL see cluster_by_MCL.
binary_cut see binary_cut.

Also the user-defined methods in all_clustering_methods can be used here.
New clustering methods can be registered by register_clustering_methods.
Please note it is better to directly call cluster_terms for clustering while not the individual cluster_by_* functions because cluster_terms does additional cluster label adjustment.

Value

A numeric vector of cluster labels (in numeric).

If catch_error is set to TRUE and if the clustering produces an error, the function returns a try-error object.

Examples

# There is no example
NULL
cmp_make_clusters

Apply various clustering methods

Description

Apply various clustering methods

Usage

cmp_make_clusters(mat, method = setdiff(all_clustering_methods(), "mclust"),
    verbose = TRUE)

Arguments

  mat   The similarity matrix.
  method Which methods to compare. All available methods are in all_clustering_methods. A value of all takes all available methods. By default mclust is excluded because its long runtime.
  verbose Whether to print messages.

Details

The function compares following default clustering methods by default:

- kmeans see cluster_by_kmeans.
- pam see cluster_by_pam.
- dynamicTreeCut see cluster_by_dynamicTreeCut.
- mclust see cluster_by_mclust. By default it is not included.
- apcluster see cluster_by_apcluster.
- hdbscan see cluster_by_hdbscan.
- fast_greedy see cluster_by_igraph.
- louvain see cluster_by_igraph.
- walktrap see cluster_by_igraph.
- MCL see cluster_by_MCL.
- binary_cut see binary_cut.

Also the user-defined methods in all_clustering_methods are also compared.

Value

A list of cluster label vectors for different clustering methods.
Examples

```r
## Not run:
mat = readRDS(system.file("extdata", "random.GO_BP_sim_mat.rds", package = "simplifyEnrichment"))
clt = cmp_make_clusters(mat)
## End(Not run)
```

### Description

Make plots for comparing clustering methods

#### Usage

```r
cmp_make_plot(mat, clt, plot_type = c("mixed", "heatmap"), nrow = 3)
```

#### Arguments

- `mat` A similarity matrix.
- `clt` A list of clusterings from `cmp_make_clusters`.
- `plot_type` What type of plots to make. See Details.
- `nrow` Number of rows of the layout when `plot_type` is set to `heatmap`.

#### Details

If `plot_type` is the default value `mixed`, a figure with three panels generated:

- A heatmap of the similarity matrix with different classifications as row annotations.
- A heatmap of the pair-wise concordance of the classifications of every two clustering methods.
- Barplots of the difference scores for each method (calculated by `difference_score`), the number of clusters (total clusters and the clusters with size >= 5) and the mean similarity of the terms that are in the same clusters.

If `plot_type` is `heatmap`. There are heatmaps for the similarity matrix under clusterings from different methods. The last panel is a table with the number of clusters under different clusterings.

#### Value

No value is returned.
**compare_clustering_methods**

*Compare clustering methods*

## Examples

```r
## Not run:
mat = readRDS(system.file("extdata", "random_GO_BP_sim_mat.rds", package = "simplifyEnrichment"))
clt = cmp_make_clusters(mat)
cmp_make_plot(mat, clt)
cmp_make_plot(mat, clt, plot_type = "heatmap")
## End(Not run)
```

## Description

Compare clustering methods

## Usage

```r
compare_clustering_methods(mat, method = setdiff(all_clustering_methods(), "mclust"),
plot_type = c("mixed", "heatmap"), nrow = 3, verbose = TRUE)
```

## Arguments

- **mat**: The similarity matrix.
- **method**: Which methods to compare. All available methods are in `all_clustering_methods`. A value of `all` takes all available methods. By default `mclust` is excluded because its long runtime.
- **plot_type**: See explanation in `cmp_make_plot`.
- **nrow**: Number of rows of the layout when `plot_type` is set to `heatmap`.
- **verbose**: Whether to print messages.

## Details

The function compares following clustering methods by default:

- kmeans see `cluster_by_kmeans`.
- pam see `cluster_by_pam`.
- dynamicTreeCut see `cluster_by_dynamicTreeCut`.
- mclust see `cluster_by_mclust`. By default it is not included.
- apcluster see `cluster_by_apcluster`.
- hdbscan see `cluster_by_hdbscan`.
- fast greedy see `cluster_by_igraph`.
louvain see `cluster_by_igraph`.
walktrap see `cluster_by_igraph`.
MCL see `cluster_by_MCL`.
binary_cut see `binary_cut`.

This function is basically a wrapper function. It calls the following two functions:

- `cmp_make_clusters`: applies clustering with different methods.
- `cmp_make_plot`: makes the plots.

Value

No value is returned.

Examples

```r
## Not run:
mat = readRDS(system.file("extdata", "random_GQ_BP_sim_mat.rds", 
    package = "simplifyEnrichment"))
compare_clustering_methods(mat)
compare_clustering_methods(mat, plot_type = "heatmap")
## End(Not run)
```

---

**count_words**

*Calculate word frequency*

Description

Calculate word frequency

Usage

```r
count_words(term,
    exclude_words = NULL, stop_words = stopwords(),
    min_word_length = 1, tokenizer = 'words', transform_case = tolower,
    remove_numbers = TRUE, remove_punctuation = TRUE, custom_transformer = NULL,
    stemming = FALSE, dictionary = NULL)
```

Arguments

- `term` A vector of description texts.
- `exclude_words` The words that should be excluded.
- `stop_words` The stop words that should be removed.
- `min_word_length` Minimum length of the word to be counted.
tokenizer  The tokenizer function, one of the values accepted by `tm::termFreq`.
transform_case The function normalizing lettercase of the words.
remove_numbers Whether to remove numbers.
remove_punctuation Whether to remove punctuation.
custom_transformer Custom function that transforms words.
stemming Whether to only keep the roots of inflected words.
dictionary A vector of words to be counted (if given all other words will be excluded).

Details


Value

A data frame with words and frequencies.

Examples

```r
gm = readRDS(system.file("extdata", "random.GO_BP_sim_mat.rds", package = "simplifyEnrichment"))
go_id = rownames(gm)
go_term = AnnotationDbi::select(GO.db::GO.db, keys = go_id, columns = "TERM")$TERM
count_words(go_term)
```

dend_node_apply  Apply functions on every node in a dendrogram

Description

Apply functions on every node in a dendrogram

Usage

`dend_node_apply(dend, fun)`

Arguments

dend  A dendrogram.
fun  A self-defined function.
Details

The function returns a vector or a list as the same length as the number of nodes in the dendrogram. The self-defined function can have one single argument which is the sub-dendrogram at a certain node. E.g. to get the number of members at every node:

```
    dend_node_apply(dend, function(d) attr(d, "members"))
```

The self-defined function can have a second argument, which is the index of current sub-dendrogram in the complete dendrogram. E.g. `dend[[1]]` is the first child node of the complete dendrogram and `dend[[c(1, 2)]]` is the second child node of `dend[[1]]`, et al. This makes that at a certain node, it is possible to get information of its child nodes and parent nodes.

```
    dend_node_apply(dend, function(d, index) {
        dend[[c(index, 1)]] # is the first child node of d, or simply d[[1]]
        dend[index[-length(index)]] # is the parent node of d
        ...
    })
```

Note for the top node, the value of `index` is `NULL`.

Value

A vector or a list, depends on whether `fun` returns a scalar or more complex values.

Examples

```
mat = matrix(rnorm(100), 10)
dend = as.dendrogram(hclust(dist(mat)))
# number of members on every node
dend_node_apply(dend, function(d) attr(d, "members"))
# the depth on every node
dend_node_apply(dend, function(d, index) length(index))
```

---

difference_score

**Difference score**

Description

Difference score

Usage

```
difference_score(mat, cl)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mat</td>
<td>The similarity matrix.</td>
</tr>
<tr>
<td>cl</td>
<td>Cluster labels.</td>
</tr>
</tbody>
</table>
**DO_similarity**

**Details**
This function measures the different between the similarity values for the terms that belong to the same clusters and in different clusters. The difference score is the Kolmogorov-Smirnov statistic between the two distributions.

**Value**
A numeric scalar.

**Examples**
```r
cl = binary_cut(mat)
difference_score(mat, cl)
```

---

**DO_similarity**  
*Calculate Disease Ontology (DO) semantic similarity matrix*

**Description**
Calculate Disease Ontology (DO) semantic similarity matrix

**Usage**
```r
DO_similarity(do_id, measure = "Rel", remove_orphan_terms = FALSE)
```

**Arguments**
- `do_id`: A vector of DO IDs.
- `measure`: Semantic measure for the DO similarity, pass to `doSim`.
- `remove_orphan_terms`: Whether to remove terms that have zero similarity to all other terms?

**Details**
This function is basically a wrapper on `doSim`.

**Value**
A symmetric matrix.

**Examples**
```r
require(DOSE)
do_id = random_DO(10)
DO_similarity(do_id)
```
Modify nodes in a dendrogram

Usage

edit_node(dend, fun = function(d, index) d)

Arguments

dend A dendrogram.
fun A self-defined function.

Details

if fun only has one argument, it is basically the same as dendrapply, but it can have a second argument which is the index of the node in the dendrogram, which makes it possible to get information of child nodes and parent nodes for a specific node.

As an example, we first assign random values to every node in the dendrogram:

```r
mat = matrix(rnorm(100), 10)
dend = as.dendrogram(hclust(dist(mat)))
dend = edit_node(dend, function(d) {attr(d, 'score') = runif(1); d})
```

Then for every node, we take the maximal absolute difference to all its child nodes and parent node as the attribute abs_diff

```r
dend = edit_node(dend, function(d, index) {
  n = length(index)
  s = attr(d, "score")
  if(is.null(index)) { # d is the top node
    s_children = sapply(d, function(x) attr(x, "score"))
    s_parent = NULL
  } else if(is.leaf(d)) { # d is the leaf
    s_children = NULL
    s_parent = attr(dend[[index[-n]]], "score")
  } else {
    s_children = sapply(d, function(x) attr(x, "score"))
    s_parent = attr(dend[[index[-n]]], "score")
  }
  abs_diff = max(abs(s - c(s_children, s_parent)))
  attr(d, "abs_diff") = abs_diff
  return(d)
})
```
export_to_shiny_app

Value

A dendrogram object.

Examples

# There is no example
NULL

export_to_shiny_app  *Interactively visualize the similarity heatmap*

Description

Interactively visualize the similarity heatmap

Usage

export_to_shiny_app(mat, cl = binary_cut(mat))

Arguments

- **mat**: A similarity matrix.
- **cl**: Cluster labels inferred from the similarity matrix, e.g. from `cluster_terms` or `binary_cut`.

Examples

if(interactive()) {
    mat = readRDS(system.file("extdata", "random_GO_BP_sim_mat.rds", package = "simplifyEnrichment"))
    cl = binary_cut(mat)
    export_to_shiny_app(mat, cl)
}

GO_similarity  *Calculate Gene Ontology (GO) semantic similarity matrix*

Description

Calculate Gene Ontology (GO) semantic similarity matrix

Usage

GO_similarity(go_id, ont = NULL, db = 'org.Hs.eg.db', measure = "Rel", remove_orphan_terms = FALSE)
Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>go_id</td>
<td>A vector of GO IDs.</td>
</tr>
<tr>
<td>ont</td>
<td>GO ontology. Value should be one of &quot;BP&quot;, &quot;CC&quot; or &quot;MF&quot;. If it is not specified, the function automatically identifies it by random sampling 10 IDs from go_id (see guess_ont).</td>
</tr>
<tr>
<td>db</td>
<td>Annotation database. It should be from <a href="https://bioconductor.org/packages/3.10/BiocViews.html#___OrgDb">https://bioconductor.org/packages/3.10/BiocViews.html#___OrgDb</a>. The value can also directly be a OrgDb object.</td>
</tr>
<tr>
<td>measure</td>
<td>Semantic measure for the GO similarity, pass to termSim.</td>
</tr>
<tr>
<td>remove_orphan_terms</td>
<td>Whether to remove terms that have zero similarity to all other terms?</td>
</tr>
</tbody>
</table>

Details

This function is basically a wrapper on termSim.

Value

A symmetric matrix.

Examples

```r
go_id = random.GO(100)
mat = GO.similarity(go_id)
```

---

**guess_ont**

*Guess the ontology of the input GO IDs*

Description

Guess the ontology of the input GO IDs

Usage

```r
guess_ont(go_id, db = 'org.Hs.eg.db')
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>go_id</td>
<td>A vector of GO IDs.</td>
</tr>
<tr>
<td>db</td>
<td>Annotation database. It should be from <a href="https://bioconductor.org/packages/3.10/BiocViews.html#___OrgDb">https://bioconductor.org/packages/3.10/BiocViews.html#___OrgDb</a>. The value can also directly be a OrgDb object.</td>
</tr>
</tbody>
</table>

Details

10 GO IDs are randomly sampled and checked.
Value

A single character scalar of "BP", "CC" or "MF". If there are more than one ontologies detected. It returns NULL.

Examples

```r
go_id = random_GO(100)
guess_ont(go_id)
```

Description

Height for word_cloud grob

Usage

```r
## S3 method for class 'word_cloud'
heightDetails(x)
```

Arguments

- `x` The word_cloud grob returned by `word_cloud_grob`.

Value

A `unit` object.

Examples

```r
# There is no example
NULL
```
ht_clusters  

Visualize the similarity matrix and the clustering

Description

Visualize the similarity matrix and the clustering

Usage

ht_clusters(mat, 
           cl, 
           dend = NULL, 
           col = c("white", "red"),

# arguments that control the word cloud annotation
draw_word_cloud = TRUE, 
min_term = round(nrow(mat)*0.01),
order_by_size = FALSE,
stat = "pvalue", 
min_stat = ifelse(stat == "count", 5, 0.05),
exclude_words = character(0),
max_words = 10,
word_cloud_grob_param = list(),
fontsize_range = c(4, 16),
bg_gp = gpar(fill = "#DDDDDD", col = "#AAAAAA"),

# arguments that control the heatmaps
column_title = NULL,
ht_list = NULL,
use_raster = TRUE,
run_draw = TRUE,
...)

Arguments

mat  
A similarity matrix.

cl  
Cluster labels inferred from the similarity matrix, e.g. from cluster_terms or binary_cut.

dend  
Used internally.

col  
A vector of colors that map from 0 to the 97.5\textsuperscript{th} percentile of the similarity values. The value can also be a color mapping function generated by colorRamp2.

draw_word_cloud  
Whether to draw the word clouds.

min_term  
Minimal number of functional terms in a cluster. All the clusters with size less than min_term are all merged into one separated cluster in the heatmap.
**keyword_enrichment_from_GO**

**Description**

Keyword enrichment for GO terms

**Value**

A `HeatmapList-class` object.

**Examples**

```r
## Not run:
mat = readRDS(system.file("extdata", "random_GO_BP_sim_mat.rds", package = "simplifyEnrichment"))
cl = binary_cut(mat)
ht_clusters(mat, cl, word_cloud_grob_param = list(max_width = 80))
ht_clusters(mat, cl, word_cloud_grob_param = list(max_width = 80), order_by_size = TRUE)
## End(Not run)
```
Usage

\texttt{keyword_enrichment_from_GO(go_id, min\_bg = 5, min\_term = 2)}

Arguments

- **go_id**: A vector of GO IDs.
- **min_bg**: Minimal number of GO terms (in the background, i.e. all GO terms in the GO database) that contain a specific keyword.
- **min_term**: Minimal number of GO terms (GO terms in \texttt{go_id}) that contain a specific keyword.

Details

The enrichment is applied by Fisher’s exact test. For a keyword, there is the following 2x2 contingency table:

- | contains the keyword | does not contain the keyword |
  | In the GO set | s11 | s12 |
  | Not in the GO set | s21 | s22 |

where s11, s12, s21 and s22 are number of GO terms in each category.

Value

A data frame with keyword enrichment results.

Examples

```r
## Not run:
go_id = random_GO(100)
keyword_enrichment_from_GO(go_id)
## End(Not run)
```

\[\text{partition\_by\_hclust} \quad \text{Partition by hclust}\]

Description

Partition by hclust

Usage

\texttt{partition\_by\_hclust(mat)}

Arguments

- **mat**: The similarity matrix.
Details

The "ward.D2" clustering method was used.
This function is used to set to the partition_fun argument in binary_cut.

Examples

# There is no example
NULL
Description

Partition by kmeans++

Usage

\texttt{partition\_by\_kmeanspp(mat)}

Arguments

\texttt{mat} \hspace{1cm} \text{The similarity matrix.}

Details

This function is used to set to the \texttt{partition\_fun} argument in \texttt{binary\_cut}.

Examples

\begin{verbatim}
# There is no example
NULL
\end{verbatim}

Description

Partition by PAM

Usage

\texttt{partition\_by\_pam(mat)}

Arguments

\texttt{mat} \hspace{1cm} \text{The similarity matrix.}

Details

The clustering is performed by \texttt{pam} with setting pamonce argument to 5.

This function is used to set to the \texttt{partition\_fun} argument in \texttt{binary\_cut}. 
plot_binary_cut

**Examples**

```r
# There is no example
NULL
```

---

**plot_binary_cut**

*Visualize the process of binary cut*

---

**Description**

Visualize the process of binary cut

**Usage**

```r
plot_binary_cut(mat, value_fun = area_above_ecdf, cutoff = 0.85, 
partiton_fun = partition_by_pam, dend = NULL, dend_width = unit(3, "cm"), 
depth = NULL, show_heatmap_legend = TRUE, ...)
```

**Arguments**

- `mat`: The similarity matrix.
- `value_fun`: A function that calculates the scores for the four submatrices on a node.
- `cutoff`: The cutoff for splitting the dendrogram.
- `partition_fun`: A function to split each node into two groups. Pre-defined functions in this package are `partition_by_kmeanspp`, `partition_by_pam` and `partition_by_hclust`.
- `dend`: A dendrogram object, used internally.
- `depth`: Depth of the recursive binary cut process.
- `dend_width`: Width of the dendrogram on the plot.
- `show_heatmap_legend`: Whether to show the heatmap legend.
- `...`: Other arguments.

**Details**

After the functions which perform clustering are executed, such as `simplifyGO` or `binary_cut`, the dendrogram is temporarily saved and `plot_binary_cut` directly uses this dendrogram.

**Examples**

```r
mat = readRDS(system.file("extdata", "random_GO_BP_sim_mat.rds", 
                           package = "simplifyEnrichment"))
plot_binary_cut(mat, depth = 1)
plot_binary_cut(mat, depth = 2)
plot_binary_cut(mat)
```
### random.DO

**Generate random Disease Ontology (DO) IDs**

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generate random Disease Ontology (DO) IDs</td>
</tr>
</tbody>
</table>

**Usage**

```r
random.DO(n) |
```

**Arguments**

<table>
<thead>
<tr>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of DO IDs.</td>
</tr>
</tbody>
</table>

**Details**

- `DO.db` package should be installed.

**Value**

- A vector of DO IDs.

**Examples**

```r
random.DO(100) |
```

---

### random.GO

**Generate random GO IDs**

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generate random GO IDs</td>
</tr>
</tbody>
</table>

**Usage**

```r
random.GO(n, ont = "BP", db = 'org.Hs.eg.db') |
```

**Arguments**

<table>
<thead>
<tr>
<th>n</th>
<th>ont</th>
<th>db</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of GO IDs.</td>
<td>GO ontology. Value should be one of &quot;BP&quot;, &quot;CC&quot; or &quot;MF&quot;.</td>
<td>Annotation database. It should be from <a href="https://bioconductor.org/packages/3.10/BiocViews.html#___OrgDb">https://bioconductor.org/packages/3.10/BiocViews.html#___OrgDb</a></td>
</tr>
</tbody>
</table>
register_clustering_methods

Value
A vector of GO IDs.

Examples
random_GO(100)

register_clustering_methods

Register new clustering methods

Description
Register new clustering methods

Usage
register_clustering_methods(...)

Arguments
... A named list of clustering functions, see Details.

Details
The user-defined functions should accept at least one argument which is the input matrix. The second optional argument should always be ... so that parameters for the clustering function can be passed by control argument from \texttt{cluster_terms}, \texttt{simplifyGO} or \texttt{simplifyEnrichment}. If users forget to add ..., it is added internally.

Please note, the user-defined function should automatically identify the optimized number of clusters.

The function should return a vector of cluster labels. Internally it is converted to numeric labels.

Value
No value is returned.

Examples

register_clustering_methods(
  # assume there are 5 groups
  random = function(mat, ...) sample(5, nrow(mat), replace = TRUE)
)
all_clustering_methods()
remove_clustering_methods("random")
remove_clustering_methods

Remove clustering methods

Description
Remove clustering methods

Usage
remove_clustering_methods(method)

Arguments
method A vector of method names.

Value
No value is returned.

Examples
# There is no example
NULL

reset_clustering_methods

Reset to default clustering methods

Description
Reset to default clustering methods

Usage
reset_clustering_methods()

Details
The default methods are:
kmeans see cluster_by_kmeans.
pam see cluster_by_pam.
dynamicTreeCut see cluster_by_dynamicTreeCut.
mclust see `cluster_by_mclust`
apcluster see `cluster_by_apcluster`
hdbscan see `cluster_by_hdbscan`
fast greedy see `cluster_by_igraph`
louvain see `cluster_by_igraph`
walktrap see `cluster_by_igraph`
MCL see `cluster_by_MCL`
binary cut see `binary_cut`

Value
No value is returned.

Examples

```r
all_clustering_methods()
remove_clustering_methods(c("kmeans", "mclust"))
all_clustering_methods()
reset_clustering_methods()
all_clustering_methods()
```

---

### scale fontsize

#### Scale font size

**Description**
Scale font size

**Usage**

```r
scale_fontsize(x, rg = c(1, 30), fs = c(4, 16))
```

**Arguments**

- `x` A numeric vector.
- `rg` The range.
- `fs` Range of the font size.

**Value**
A numeric vector.

**Details**
It is a linear interpolation.
select_cutoff

Select the cutoff for binary cut

Description

Select the cutoff for binary cut

Usage

select_cutoff(mat, cutoff = seq(0.6, 0.98, by = 0.01), verbose = TRUE, ...)

Arguments

mat A similarity matrix.
cutoff A list of cutoffs to test. Note the range of the cutoff values should be inside [0.5, 1].
verbose Whether to print messages.
... Pass to binary_cut.

Details

Binary cut is applied to each of the cutoff and the clustering results are evaluated by following metrics:

- difference score, calculated by difference_score.
- number of clusters.
- block mean, which is the mean similarity in the blocks in the diagonal of the heatmap.

Examples

mat = readRDS(system.file("extdata", "random_GO_BP_sim_mat.rds", package = "simplifyEnrichment"))
select_cutoff(mat)
se_opt

Global parameters

Description
Global parameters

Usage
se_opt(..., RESET = FALSE, READ.ONLY = NULL, LOCAL = FALSE, ADD = FALSE)

Arguments

Arguments for the parameters, see "details" section.
RESET Whether to reset to default values.
READ.ONLY Please ignore.
LOCAL Please ignore.
ADD Please ignore.

Details
There are the following global options:

verbose Whether to print messages.

Examples

# There is no example
NULL

simplifyEnrichment
Simplify functional enrichment results

Description
Simplify functional enrichment results

Usage
simplifyEnrichment(mat, method = "binary_cut", control = list(),
plot = TRUE, verbose = TRUE,
column_title = qq("@nrow(mat)} terms clustered by '@{method}'"),
ht_list = NULL, ...)


simplifyGO

Arguments

mat A similarity matrix.
method Method for clustering the matrix. See cluster_terms.
control A list of parameters for controlling the clustering method, passed to cluster_terms.
plot Whether to make the heatmap.
column_title Column title for the heatmap.
verbose Whether to print messages.
ht_list A list of additional heatmaps added to the left of the similarity heatmap.
... Arguments passed to ht_clusters.

Details

The usage is the same as simplifyGO.

Examples

# There is no example
NULL

simplifyGO

Simplify Gene Ontology (GO) enrichment results

Description

Simplify Gene Ontology (GO) enrichment results

Usage

simplifyGO(mat, method = "binary_cut", control = list(),
plot = TRUE, verbose = TRUE,
column_title = qq("@{nrow(mat)} GO terms clustered by '{method}'"),
ht_list = NULL, ...)

Arguments

mat A GO similarity matrix.
method Method for clustering the matrix. See cluster_terms.
control A list of parameters for controlling the clustering method, passed to cluster_terms.
plot Whether to make the heatmap.
column_title Column title for the heatmap.
verbose Whether to print messages.
ht_list A list of additional heatmaps added to the left of the similarity heatmap.
... Arguments passed to ht_clusters.
Details

This is basically a wrapper function that it first runs `cluster_terms` to cluster GO terms and then runs `ht_clusters` to visualize the clustering.

The arguments in `simplifyGO` passed to `ht_clusters` are:

- `draw_word_cloud`  Whether to draw the word clouds.
- `min_term`  Minimal number of GO terms in a cluster. All the clusters with size less than `min_term` are all merged into one single cluster in the heatmap.
- `order_by_size`  Whether to reorder GO clusters by their sizes. The cluster that is merged from small clusters (size < `min_term`) is always put to the bottom of the heatmap.
- `stat`  What values of keywords are used to map to font sizes in the word clouds.
- `exclude_words`  Words that are excluded in the word cloud.
- `max_words`  Maximal number of words visualized in the word cloud.
- `word_cloud_grob_param`  A list of graphic parameters passed to `word_cloud_grob`.
- `fontsize_range`  The range of the font size. The value should be a numeric vector with length two. The minimal font size is mapped to word frequency value of 1 and the maximal font size is mapped to the maximal word frequency. The font size interloation is linear.
- `bg_gp`  Graphic parameters for controlling the background of word cloud annotations.

Value

A data frame with two columns: GO IDs and cluster labels.

See Also

- `simplifyGOFromMultipleLists` which performs simplifyGO analysis with multiple lists of GO IDs.

Examples

```r
set.seed(123)
go_id = random_GO(500)
mat = GO_similarity(go_id)
df = simplifyGO(mat, word_cloud_grob_param = list(max_width = 80))
head(df)
```
Perform simplifyGO analysis with multiple lists of GO IDs

**Description**

Perform simplifyGO analysis with multiple lists of GO IDs

**Usage**

```r
simplifyGOFromMultipleLists(lt, go_id_column = NULL, 
  padj_column = NULL, padj_cutoff = 1e-2, 
  filter = function(x) any(x < padj_cutoff), default = 1, 
  ont = NULL, db = 'org.Hs.eg.db', measure = "Rel", 
  heatmap_param = list(NULL), show_barplot = TRUE, 
  method = "binary_cut", control = list(), 
  min_term = NULL, verbose = TRUE, column_title = NULL, ...)
```

**Arguments**

- `lt` A data frame, a list of numeric vectors (e.g. adjusted p-values) where each numeric vector has GO IDs as names, or a list of GO IDs.
- `go_id_column` Column index of GO ID if `lt` contains a list of data frames.
- `padj_column` Column index of adjusted p-values if `lt` contains a list of data frames.
- `padj_cutoff` Cut off for adjusted p-values
- `filter` A self-defined function for filtering GO IDs. By default it requires GO IDs should be significant in at least one list.
- `default` The default value for the adjusted p-values. See Details.
- `ont` GO ontology. Value should be one of "BP", "CC" or "MF". If it is not specified, the function automatically identifies it by random sampling 10 IDs from `go_id` (see `guess_ont`).
- `db` Annotation database. It should be from https://bioconductor.org/packages/3.10/BiocViews.html#___OrgDb
- `measure` Semantic measure for the GO similarity, pass to `termSim`.
- `heatmap_param` Parameters for controlling the heatmap, see Details.
- `show_barplot` Whether draw barplots which shows numbers of significant GO terms in clusters.
- `method` Pass to `simplifyGO`.
- `control` Pass to `simplifyGO`.
- `min_term` Pass to `simplifyGO`.
- `verbose` Pass to `simplifyGO`.
- `column_title` Pass to `simplifyGO`.
- `...` Pass to `simplifyGO`.
Details

The input data can have three types of formats:

• A list of numeric vectors of adjusted p-values where each vector has the GO IDs as names.

• A data frame. The column of the GO IDs can be specified with `go_id_column` argument and the column of the adjusted p-values can be specified with `padj_column` argument. If these columns are not specified, they are automatically identified. The GO ID column is found by checking whether a column contains all GO IDs. The adjusted p-value column is found by comparing the column names of the data frame to see whether it might be a column for adjusted p-values. These two columns are used to construct a numeric vector with GO IDs as names.

• A list of character vectors of GO IDs. In this case, each character vector is changed to a numeric vector where all values take 1 and the original GO IDs are used as names of the vector.

Now let's assume there are n GO lists, we first construct a global matrix where columns correspond to the n GO lists and rows correspond to the "union" of all GO IDs in the lists. The value for the ith GO ID and in the jth list are taken from the corresponding numeric vector in lt. If the jth vector in lt does not contain the ith GO ID, the value defined by default argument is taken there (e.g. in most cases the numeric values are adjusted p-values, default is set to 1). Let's call this matrix as $M_0$.

Next step is to filter $M_0$ so that we only take a subset of GO IDs of interest. We define a proper function via argument `filter` to remove GO IDs that are not important for the analysis. Functions for filter is applied to every row in $M_0$ and filter function needs to return a logical value to decide whether to remove the current GO ID. For example, if the values in lt are adjusted p-values, the filter function can be set as `function(x) any(x < padj_cutoff)` so that the GO ID is kept as long as it is significant in at least one list. After the filter, let's call the filtered matrix $M_1$.

GO IDs in $M_1$ (row names of $M_1$) are used for clustering. A heatmap of $M_1$ is attached to the left of the GO similarity heatmap so that the group-specific (or list-specific) patterns can be easily observed and corresponded to GO functions.

Argument `heatmap_param` controls several parameters for heatmap $M_1$:

• `transform`: A self-defined function to transform the data for heatmap visualization. The most typical case is to transform adjusted p-values by $-\log_{10}(x)$.

• `breaks`: break values for color interpolation.

• `col`: The corresponding values for breaks.

• `labels`: The corresponding labels.

• `name`: Legend title.

Examples

```r
# perform functional enrichment on the signatures genes from cola analysis
require(cola)
data(golub_cola)
res = golub_cola["ATC:skmeans"]
require(hu6800.db)
```
x = hu6800ENTREZID
mapped_probes = mappedkeys(x)
id_mapping = unlist(as.list(x[mapped_probes]))
lt = functional_enrichment(res, k = 3, id_mapping = id_mapping) # you can check the value of 'lt'

# a list of data frames
simplifyGOFromMultipleLists(lt, padj_cutoff = 0.001)

# a list of numeric values
lt2 = lapply(lt, function(x) structure(x$p.adjust, names = x$ID))
simplifyGOFromMultipleLists(lt2, padj_cutoff = 0.001)

# a list of GO IDS
lt3 = lapply(lt, function(x) x$ID[x$p.adjust < 0.001])
simplifyGOFromMultipleLists(lt3)

---

subset_enrichResult  Subset method of the enrichResult class

Description

Subset method of the enrichResult class

Usage

subset_enrichResult(x, i)

Arguments

x  A enrichResult object from 'clusterProfiler' or other related packages.
i  Row indices.

Value

Still a enrichResult object but with the selected subset of rows.

Examples

# There is no example
NULL
**summarizeGO**

A simplified way to visualize enrichment in GO clusters

### Description

A simplified way to visualize enrichment in GO clusters

### Usage

```r
summarizeGO(go_id, value = NULL, aggregate = mean, method = "binary_cut", control = list(), verbose = TRUE, axis_label = "Value", title = ", legend_title = axis_label,

min_term = round(nrow(mat)*0.01),
stat = "pvalue",
min_stat = ifelse(stat == "count", 5, 0.05),
exclude_words = character(0),
max_words = 6,
word_cloud_grob_param = list(),
fontsize_range = c(4, 16),
bg_gp = gpar(fill = "#DDDDDD", col = "#AAAAAA")
)
```

### Arguments

- **go_id**: A vector of GO IDs.
- **value**: A list of numeric value associate with `go_id`. We suggest to use -log10(p.adjust) or -log2(fold enrichment) as the values.
- **aggregate**: Function to aggregate values in each GO cluster.
- **method**: Method for clustering the matrix. See `cluster_terms`.
- **control**: A list of parameters for controlling the clustering method, passed to `cluster_terms`.
- **verbose**: Whether to print messages.
- **axis_label**: X-axis label.
- **title**: Title for the whole plot.
- **legend_title**: Title for the legend.
- **min_term**: Minimal number of functional terms in a cluster. All the clusters with size less than `min_term` are all merged into one separated cluster in the heatmap.
- **stat**: Type of value for mapping to the font size of keywords in the word clouds. There are two options: "count": simply number of keywords; "pvalue": enrichment on keywords is performed (by fisher's exact test) and -log10(pvalue) is used to map to font sizes.
- **min_stat**: Minimal value for `stat` for selecting keywords.
- **exclude_words**: Words that are excluded in the word cloud.
term_similarity

max_words Maximal number of words visualized in the word cloud.

word_cloud_grob_param A list of graphic parameters passed to word_cloud_grob.

fontsize_range The range of the font size. The value should be a numeric vector with length two. The font size interpolation is linear.

bg_gp Graphics parameters for controlling word cloud annotation background.

Details

There are several other ways to specify GO IDs and the associated values.

1. specify value as a named vector where GO IDs are the names.
2. specify value as a list of numeric named vectors. In this case, value contains multiple enrichment results.

Examples

# There is no example
NULL

Description

Similarity between terms based on the overlap of genes

Usage

term_similarity(gl, method = c("kappa", "jaccard", "dice", "overlap"), all = NULL, remove_negative = TRUE)

Arguments

- gl: A list of genes that are in the terms.
- method: The similarity measurement.
- all: The universe set.
- remove_negative: If the value is negative, reset to zero

Details

The definition of the four similarity measurements can be found at https://jokergoo.github.io/simplifyEnrichment_supplementary/supplS01_coefficient_definition/supplS01_coefficient_definition.html.

Value

A symmetric matrix.
**term_similarity_from_enrichResult**

_Similarity between terms in the enrichResult class_

**Description**

Similarity between terms in the enrichResult class

**Usage**

```r
term_similarity_from_enrichResult(x, ...)
```

**Arguments**

- `x` A `enrichResult` object from 'clusterProfiler' or other related packages.
- `...` Pass to `term_similarity`.

**Details**

The object is normally from the 'clusterProfiler', 'DOSE', 'meshes' or 'ReactomePA' package.

**Value**

A symmetric matrix.

**Examples**

```r
# There is no example
NULL
```
term_similarity_from_gmt

Similarity between terms from a gmt file

Description

Similarity between terms from a gmt file

Usage

term_similarity_from_gmt(term_id, gmt, extract_term_id = NULL, ...)

Arguments

term_id: A vector of terms.
gmt: The path of the gmt file.
extract_term_id: If the term ID in the first column only as a substring, setting a function to extract this substring.
...

Pass to term_similarity.

Value

A symmetric matrix.

Examples

# There is no example
NULL

term_similarity_from_KEGG

Similarity between KEGG terms

Description

Similarity between KEGG terms

Usage

term_similarity_from_KEGG(term_id, ...)

**term_similarity_from_MSigDB**

*Similarity between MSigDB terms*

**Description**

Similarity between MSigDB terms

**Usage**

```r
term_similarity_from_MSigDB(term_id, category = NULL, subcategory = NULL, ...)
```

**Arguments**

- `term_id`: A vector of MSigDB gene set names.
- `category`: E.g., 'C1', 'C2', pass to `msigdbr`.
- `subcategory`: E.g., 'CGP', 'BP', pass to `msigdbr`.
- `...`: Pass to `term_similarity`.

**Value**

A symmetric matrix.

**Examples**

```r
# There is no example
NULL
```
term_similarity_from_Reactome

*Similarity between Reactome terms*

**Description**

Similarity between Reactome terms

**Usage**

`term_similarity_from_Reactome(term_id, ...)`

**Arguments**

- `term_id`: A vector of Reactome IDs.
- `...`: Pass to `term_similarity`.

**Value**

A symmetric matrix.

**Examples**

```r
# There is no example
NULL
```

---

widthDetails.word_cloud

*Width for word_cloud grob*

**Description**

Width for word_cloud grob

**Usage**

```r
## S3 method for class 'word_cloud'
widthDetails(x)
```

**Arguments**

- `x`: The word_cloud grob returned by `word_cloud_grob`.

**Value**

A `unit` object.
Examples

# There is no example
NULL

word_cloud_grob  A simple grob for the word cloud

Description

A simple grob for the word cloud

Usage

word_cloud_grob(text, fontsize,
    line_space = unit(4, "pt"), word_space = unit(4, "pt"), max_width = unit(80, "mm"),
    col = function(fs) circlize::rand_color(length(fs), luminosity = "dark"),
    add_new_line = FALSE, test = FALSE)

Arguments

text  A vector of words.

fontsize  The corresponding font size. With the frequency of the words known, scale_fontsize
            can be used to linearly interpolate frequencies to font sizes.

line_space  Space between lines. The value can be a unit object or a numeric scalar which
            is measured in mm.

word_space  Space between words. The value can be a unit object or a numeric scalar which
            is measured in mm.

max_width  The maximal width of the viewport to put the word cloud. The value can be a
            unit object or a numeric scalar which is measured in mm. Note this might be
            larger than the final width of the returned grob object.

col  Colors for the words. The value can be a vector, in numeric or character, which
            should have the same length as text. Or it is a self-defined function that takes
            the font size vector as the only argument. The function should return a color
            vector. See Examples.

add_new_line  Whether to add new line after every word? If TRUE, each word will be in a
            separated line.

test  Internally used. It basically adds borders to the words and the viewport.

Value

A grob object. The width and height of the grob can be get by grobWidth and grobHeight.
Examples

# very old R versions do not have strrep() function
if(!exists("strrep")) {
  strrep = function(x, i) paste(rep(x, i), collapse = "")
}

words = sapply(1:30, function(x) strrep(sample(letters, 1), sample(3:10, 1)))
require(grid)

gb = word_cloud_grob(words, fontsize = runif(30, min = 5, max = 30),
  max_width = 100)
grid.newpage(); grid.draw(gb)

# color as a single scalar
gb = word_cloud_grob(words, fontsize = runif(30, min = 5, max = 30),
  max_width = 100, col = 1)
grid.newpage(); grid.draw(gb)

# color as a vector
gb = word_cloud_grob(words, fontsize = runif(30, min = 5, max = 30),
  max_width = 100, col = 1:30)
grid.newpage(); grid.draw(gb)

# color as a function
require(circlize)
col_fun = colorRamp2(c(5, 17, 30), c("blue", "black", "red"))

gb = word_cloud_grob(words, fontsize = runif(30, min = 5, max = 30),
  max_width = 100, col = function(fs) col_fun(fs))
grid.newpage(); grid.draw(gb)
Index

all_clustering_methods, 3, 14, 15, 17
anno_empty, 5
anno_link, 4, 5
anno_word_cloud, 4, 6
anno_word_cloud_from_GO, 5
apcluster, 8
area_above_ecdf, 6
binary_cut, 3, 7, 14, 15, 18, 23, 26, 29–31, 35, 36
cluster_by_apcluster, 3, 8, 14, 15, 17, 35
cluster_by_dynamicTreeCut, 3, 8, 14, 15, 17, 34
cluster_by_hdbscan, 3, 9, 14, 15, 17, 35
cluster_by_igraph, 3, 10, 14, 15, 17, 18, 35
cluster_by_kmeans, 3, 11, 14, 15, 17, 34
cluster_by_MCL, 3, 11, 14, 15, 18, 35
cluster_by_mclust, 3, 12, 14, 15, 17, 35
cluster_by_pam, 13, 14, 15, 17, 34
cluster_fast_greedy, 10
cluster_leading_eigen, 10
colorRamp2, 26
cmp_make_clusters, 15, 16, 18
cmp_make_plot, 16, 17, 18
difference_score, 16, 20, 36
dend_node_apply, 19
dendrapply, 22
doSim, 21
detectDO, 32
detectGO, 32
difference_score, 16, 20, 36
D0_similarity, 21
detectDO, 32
detectGO, 32
diffScore, 36
diffSimilarity, 20, 36
diff_similarity, 20, 36
diffSimilarity, 20, 36
diff_similarity, 20, 36
detectDOFromMultipleLists, 39, 40
detectGOFromMultipleLists, 39, 40
detectGO, 32
detectDO, 32
diffSimilarity, 20, 36
diff_similarity, 20, 36
diffSimilarity, 20, 36
diff_similarity, 20, 36
detectDOFromMultipleLists, 39, 40
detectGOFromMultipleLists, 39, 40
detectGO, 32
detectDO, 32
diffSimilarity, 20, 36
diff_similarity, 20, 36
diffSimilarity, 20, 36
diff_similarity, 20, 36
detectDOFromMultipleLists, 39, 40
detectGOFromMultipleLists, 39, 40
detectGO, 32
export_to_shiny_app, 23
 GO_similarity, 23
 grob, 49
 grobHeight, 49
 grobWidth, 49
 guess_ont, 24, 24, 40
 heightDetails.word_cloud, 25
 ht_clusters, 26, 38, 39
 keyword_enrichment_from_GO, 27
 kmeans, 11
 mcl, 12
 Mclust, 12
 msigdb, 47
 pam, 30
 pamk, 13
 partition_by_hclust, 7, 28, 31
 partition_by_kmeans, 29
 partition_by_kmeanspp, 7, 30, 31
 partition_by_pam, 7, 30, 31
 plot_binary_cut, 31, 31
 random_DO, 32
 random_GO, 32
 register_clustering_methods, 4, 14, 33
 remove_clustering_methods, 34
 reset_clustering_methods, 34
 scale_fontsize, 35, 49
 se_opt, 37
 select_cutoff, 36
 simplifyEnrichment, 33, 37
 simplifyGO, 31, 33, 38, 38, 39, 40
 simplifyGOFromMultipleLists, 39, 40
 subset_enrichResult, 42
 se_opt, 37
 simplifyEnrichment, 33, 37
 simplifyGO, 31, 33, 38, 38, 39, 40
 simplifyGOFromMultipleLists, 39, 40
 subset_enrichResult, 42
 term_similarity, 44, 45–48
term_similarity_from_enrichResult, 45
term_similarity_from_gmt, 46
term_similarity_from_KEGG, 46
term_similarity_from_MSigDB, 47
term_similarity_from_Reactome, 48
termSim, 24, 40
unit, 25, 48, 49
widthDetails.word_cloud, 48
word_cloud_grob, 4, 25, 27, 39, 44, 48, 49