Package 'simplifyEnrichment'

March 7, 2021

Type Package

Title Simplify Functional Enrichment Results

Version 1.0.0

Date 2020-10-07

Author Zuguang Gu

Maintainer Zuguang Gu <z.gu@dkfz.de>

Depends R (>= 3.6.0), BiocGenerics, grid

Imports GOSemSim, ComplexHeatmap (>= 2.5.4), circlize, GetoptLong, digest, tm, GO.db, org.Hs.eg.db, AnnotationDbi, slam, methods, clue, grDevices, graphics, stats, utils, proxyC, Matrix, cluster (>= 1.14.2)

Suggests knitr, ggplot2, cowplot, mclust, apcluster, MCL, dbscan, igraph, gridExtra, dynamicTreeCut, testthat, gridGraphics, clusterProfiler, msigdbr, DOSE, DO.db, reactome.db, flexclust, BiocManager

Description A new method (binary cut) is proposed to effectively cluster GO terms into groups from the semantic similarity matrix. Summaries of GO terms in each cluster are visualized by word clouds.

biocViews Software, Visualization, GO, Clustering, GeneSetEnrichment

URL https://github.com/jokergoo/simplifyEnrichment, https://simplifyEnrichment.github.io

VignetteBuilder knitr

License MIT + file LICENSE

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

git_branch RELEASE_3_12

git_last_commit 8e81880

git_last_commit_date 2020-10-27

Date/Publication 2021-03-06

R topics documented:

`all_clustering_methods` *All clustering methods*

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`.

`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`.

Value

A vector of method names.

See Also

New methods can be added by `register_clustering_methods`.

Examples

```
all_clustering_methods()
```

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

Description

Cluster functional terms by recursively binary cutting the similarity matrix

Usage

```
binary_cut(mat, value_fun = median, partition_fun = partition_by_pam,
    cutoff = 0.85, cache = FALSE, try_all_partition_fun = FALSE)
```

Arguments

mat	A similarity matrix.
value_fun	Value function to calculate the score for each node in 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.
cutoff	The cutoff for splitting the dendrogram.
cache	Whether the dendrogram should be cached. Internally used.
try_all_partition_fun	Different partition_fun gives different clusterings. If the vaule 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.

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, ...)
```

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("cluster_fast_greedy",
    "cluster_leading_eigen",
    "cluster_louvain",
    "cluster_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 hte 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:

`cluster_fast_greedy` uses `cluster_fast_greedy`.

`cluster_leading_eigen` uses `cluster_leading_eigen`.

`cluster_louvain` uses `cluster_louvain`.

`cluster_walktrap` uses `cluster_walktrap`.

Value

A vector of cluster labels (in numeric).

Examples

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

`cluster_by_kmeans` *Cluster similarity matrix by k-means clustering*

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
```

cluster_by_MCL *Cluster similarity matrix by MCL*

Description

Cluster similarity matrix by MCL

Usage

```
cluster_by_MCL(mat, addLoops = FALSE, ...)
```

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_terms` *Cluster functional terms*

Description

Cluster functional terms

Usage

```
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`.

`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`.

`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`.

`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`.

`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` are also compared.

Value

A list of cluster label vectors for different clustering methods.

Examples

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

## End(Not run)
```

---

cmp_make_plot *Make plots for comparing clustering methods*

Description

Make plots for comparing clustering methods

Usage

```
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.

Examples

```
## 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)
```

`compare_clustering_methods` *Compare clustering methods*

Description

Compare clustering methods

Usage

```
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`.

`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`.

`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.

This functon 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

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

## End(Not run)
```

---

count_word *Calculate word frequency*

Description

Calculate word frequency

Usage

```
count_word(go_id, term = NULL, exclude_words = NULL)
```

Arguments

go_id	A vector of GO IDs.
term	The corresponding names or description of terms if the input are not GO terms.
exclude_words	The words that should be excluded.

Details

The input can be simply set with a vector of GO id to go_id argument that the GO names are automatically extracted. If the input are not GO terms, users need to provide a vector of long names/descriptions by term argument.

If the input is GO id, the following words are excluded: c("via","protein","factor","side","type","specific") They are analyzed by simplifyEnrichment:::all_GO_word_count().

The text preprocessing followings the instructions from http://www.sthda.com/english/wiki/word-cloud-generator-in-r-one-killer-function-to-do-everything-you-need .

Value

A data frame with words and frequencies.

Examples

```
go_id = random_GO(500)
head(count_word(go_id))
```

---

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

mat	The similarity matrix.
cl	Cluster labels.

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

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

```
DO_similarity(do_id, measure = "Rel")
```

Arguments

do_id	A vector of DO IDs.
measure	Semantic measurement for the DO similarity, pass to doSim.

Details

This function is basically a wrapper on doSim.

Value

A symmetric matrix.

Examples

```
require(DOSE)
do_id = random_DO(10)
DO_similarity(do_id)
```

edit_node *Modify nodes in a dendrogram*

Description

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:

```
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

```
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)
})
```

Value

A dendrogram object.

Examples

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

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

Description

Calculate Gene Ontology (GO) semantic similarity matrix

Usage

```
GO_similarity(go_id, ont, db = 'org.Hs.eg.db', measure = "Rel")
```

Arguments

go_id	A vector of GO IDs.
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 measurement for the GO similarity, pass to termSim.

Details

This function is basically a wrapper on termSim.

Value

A symmetric matrix.

Examples

```
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

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

Arguments

go_id	A vector of GO IDs.
db	Annotation database. It should be from https://bioconductor.org/packages/3.10/BiocViews.html#___OrgDb

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

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

---

heightDetails.word_cloud *Height for word_cloud grob*

Description

Height for word_cloud grob

Usage

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

Arguments

x	The word_cloud grob returned by word_cloud_grob.

Value

A unit object.

Examples

```
# 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"),
    draw_word_cloud = is_GO_id(rownames(mat)[1]) || !is.null(term),
    term = NULL, min_term = 5,
    order_by_size = FALSE, cluster_slices = FALSE,
    exclude_words = character(0), max_words = 10,
    word_cloud_grob_param = list(), fontsize_range = c(4, 16),
    column_title = NULL, ht_list = NULL, use_raster = 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 95^th percentile of the similarity values.
draw_word_cloud	Whether to draw the word clouds.
term	The full name or the description of the corresponding GO IDs.
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.
order_by_size	Whether to reorder clusters by their sizes. The cluster that is merged from small clusters (size < min_term) is always put to the bottom of the heatmap.
cluster_slices	Whether to cluster slices.
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 interlopation is linear.
column_title	Column title for the heatmap.
ht_list	A list of additional heatmaps added to the left of the similarity heatmap.
use_raster	Whether to write the heatmap as a raster image.
...	Other arguments passed to draw,HeatmapList-method.

Value

A HeatmapList-class object.

Examples

```
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)
```

partition_by_hclust *Partition by hclust*

Description

Partition by hclust

Usage

```
partition_by_hclust(mat)
```

Arguments

mat	The similarity matrix.

Details

The "ward.D2" clusering method was used.

This function is used to set to the partition_fun argument in binary_cut.

Examples

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

partition_by_kmeans *Partition by kmeans*

Description

Partition by kmeans

Usage

```
partition_by_kmeans(mat, n_repeats = 10)
```

Arguments

mat	The similarity matrix.
n_repeats	Number of repeated runs of k-means.

Details

Since k-means clustering brings randomness, this function performs k-means clustering several times and uses the final consensus partitioning.

This function is used to set to the partition_fun argument in binary_cut.

Examples

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

`partition_by_kmeanspp` *Partition by kmeans++*

Description

Partition by kmeans++

Usage

```
partition_by_kmeanspp(mat)
```

Arguments

`mat`	The similarity matrix.

Details

This function is used to set to the `partition_fun` argument in `binary_cut`.

Examples

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

`partition_by_pam` *Partition by PAM*

Description

Partition by PAM

Usage

```
partition_by_pam(mat)
```

Arguments

`mat`	The similarity matrix.

Details

The clustering is performed by `pam` with setting `pamonce` argument to 5.

This function is used to set to the `partition_fun` argument in `binary_cut`.

Examples

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

`plot_binary_cut` *Visualize the process of binary cut*

Description

Visualize the process of binary cut

Usage

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

Arguments

`mat`	The similarity matrix.
`value_fun`	Value function to calculate the score for each node in the dendrogram.
`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.
`show_heatmap_legend`	Whether to show the heatmap legend.
`...`	Other arguments.

Details

After the functions which performs clustering are executed, such as `simplifyGO` or `binary_cut`, the dendrogram is temporarily saved and `plot_binary_cut` directly uses this dendrogram. So, if the partition function brings randomness, it makes sure the clustering is the same as the one made by e.g. `simplifyGO`.

Examples

```
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*

Description

Generate random Disease Ontology (DO) IDs

Usage

```
random_DO(n)
```

Arguments

n	Number of DO IDs.

Details

DO.db package should be installed.

Value

A vector of DO IDs.

Examples

```
random_DO(100)
```

random_GO *Generate random GO IDs*

Description

Generate random GO IDs

Usage

```
random_GO(n, ont = "BP", db = 'org.Hs.eg.db')
```

Arguments

n	Number of GO IDs.
ont	GO ontology. Value should be one of "BP", "CC" or "MF".
db	Annotation database. It should be from https://bioconductor.org/packages/3.10/BiocViews.html#___OrgDb

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 `cluster_terms`, `simplifyGO` or `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.

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.

`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`.

Value

No value is returned.

Examples

```
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

```
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.

Detaisl

It is a linear interpolation.

Examples

```
x = runif(10, min = 1, max = 20)
# scale x to fontsize 4 to 16.
scale_fontsize(x)
```

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)
```

simplifyEnrichment *Simplify functional enrichment results*

Description

Simplify functional enrichment results

Usage

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

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.
term	The full name or the description of the corresponding terms.
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, except you need to manually provide the term names by term argument if you want to draw the word clouds.

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, term = NULL, 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.
term	The full name or the description of the corresponding GO IDs. The values are automatically extracted if it is not provided.
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.

`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 interlopation is linear.

Value

A data frame with three columns: GO IDs, GO term names and cluster labels.

Examples

```
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)
```

---

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
```

`term_similarity` *Similarity between terms based on the overlap of genes*

Description

Similarity between terms based on the overlap of genes

Usage

```
term_similarity(gl, method = c("kappa", "jaccard", "dice", "overlap"))
```

Arguments

`gl`	A list of genes that are in the terms.
`method`	The similarity measurement.

Details

The definition of the four similarity measurements can be found at https://simplifyenrichment.github.io/supplementary/suppl1_coefficient_definition/suppl1_coefficient_definition.html .

Value

A symmetric matrix.

Examples

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

`term_similarity_from_enrichResult` *Similarity between terms in the enrichResult class*

Description

Similarity between terms in the enrichResult class

Usage

```
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

```
# 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 contained 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, ...)
```

Arguments

term_id	A vector of KEGG IDs, e.g., hsa001.
...	Pass to term_similarity.

Value

A symmetric matrix.

Examples

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

term_similarity_from_MSigDB

Similarity between MSigDB terms

Description

Similarity between MSigDB terms

Usage

```
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

```
# 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

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

`widthDetails.word_cloud` *Width for word_cloud grob*

Description

Width for word_cloud grob

Usage

```
## 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"),
    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.
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