Package: goat (via r-universe)

August 21, 2024

Type Package

Title Gene Set Analysis Using the Gene Set Ordinal Association Test

Version 1.0

Description Perform gene set enrichment analyses using the Gene set Ordinal Association Test (GOAT) algorithm and visualize your results. Koopmans, F. (2024) <doi:10.1038/s42003-024-06454-5>.

URL https://github.com/ftwkoopmans/goat/

License Apache License (>= 2)

Depends R (>= 4.1.0), dplyr (>= 1.0.3)

Imports tibble (>= 3.0.0), tidyselect (>= 1.2.0), tidyr (>= 1.1.2), data.table (>= 1.14.0), Matrix (>= 1.4-0), readxl (>= 1.4.1), writexl (>= 1.4.1), Rcpp (>= 1.0.9), vctrs (>= 0.3.8), MonoPoly (>= 0.3-10), ggplot2 (>= 3.3.0), pheatmap (>= 1.0.8), treemap (>= 2.4), igraph (>= 1.2.5), ggraph (>= 2.0.0)

Suggests AnnotationDbi, GO.db, org.Hs.eg.db, fgsea, testthat (>= 3.0.0)

LinkingTo Rcpp

Encoding UTF-8

LazyData true

LazyDataCompression xz

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.1

Config/testthat/edition 3

Language en-US

Remotes github::ctlab/fgsea

Repository https://ftwkoopmans.r-universe.dev

RemoteUrl https://github.com/ftwkoopmans/goat

RemoteRef HEAD

RemoteSha 80accdfe5d500322322b5f6301f14827922fd7f9

Contents

cluster_genesets *cluster significant genesets from* test_genesets() *by geneset similarity (separately for each 'geneset source')*

Description

cluster significant genesets from `test_genesets()` by geneset similarity (separately for each 'geneset source')

Usage

```
cluster_genesets(x, genelist, hclust_method = "ward.D2")
```

Arguments

x	results from `test_genesets()`
genelist	should be the same as provided to `test_genesets()`
hclust_method	hierarchical clustering method, any of; 'ward.D', 'ward.D2' (default), 'single', 'complete', 'average'

Value

a list with elements genesets (param x), similarity, hc_row, hc_col

darken_color *naively darken a color by mixing in black*

Description

naively darken a color by mixing in black

Usage

```
darken_color(color, frac = 0.1)
```

Arguments

color	input colors
frac	fraction of black; >0 and <1

Value

adjusted value for input `color`

`download_genesets_goatrepo` *Download and parse geneset collections from the GOAT GitHub repository*

Description

while the Bioconductor respository is extensive, contains data for many species and is a part of a larger infrastructure, it might contain outdated GO data when the user is not using the latest R version. If users are on an R version that is a few years old, so will the GO data from Bioconductor be.

As an alternative, we store gene2go data from NCBI (for Human genes only!) at the GOAT GitHub repository. This function allows for a convenient way to download this data and then parse the genesets.

Alternatively you can browse the file in the data branch of the GOAT GitHub repository and download these files manually, then load them via the GOAT R function `load_genesets_go_fromfile()`.

To view all available data you can open this URL in a browser; https://github.com/ftwkoopmans/goat/tree/data

New data is automatically added biannually. The first available version is 2024-01-01, the next 2024-06-01, then 2025-01-01, and so on.

Usage

```
download_genesets_goatrepo(
  output_dir,
  type = "GO",
  version = "2024-01-01",
  ignore_cache = FALSE
)
```

Arguments

`output_dir`	full path to the directory where the downloaded files should be stored. Directory is created if it does not exist. e.g. `output_dir="~/data"` on unix systems, `output_dir="C:/data"` on Windows, or set to `output_dir=getwd()` to write output to the current working directory
`type`	the type of genesets to download. Currently, only "GO" is supported (default)
`version`	the dataset version. This must be a date in format YYYY-MM-DD. Example: "2024-01-01" (default). View all available versions at https://github.com/ftwkoopmans/goat/tree/data
`ignore_cache`	boolean, set to TRUE to force re-download and ignore cached data, if any. Default: FALSE

Value

result from respective geneset parser function. e.g. if parameter type was set to"GO" (default), this function returns the result of load_genesets_go_fromfile(). These data returned by this function is typically used as input for filter_genesets(), c.f. full example at documentation for test_genesets()

Examples

```
# note: this example will download 2 files of approx 10MB in total

# store the downloaded files in the following directory. Here, the temporary file
# directory is used. Alternatively, consider storing this data in a more permanent location.
# e.g. output_dir="~/data/go" on unix systems or output_dir="C:/data/go" on Windows
output_dir = tempdir()

# download data files with GO annotations, version 2024-01-01 (default parameter)
# these are then parsed with the load_genesets_go_fromfile() function
# if the files are already available at output_dir, the download step is skipped
genesets_asis = download_genesets_goatrepo(output_dir)

### for a basic example on how to use the data obtain here,
### refer to the example included at function documentation of: test_genesets()
```

---

download_goat_manuscript_data

Download the datasets that were used in the GOAT manuscript

Description

Downloads OMICs-based datasets that were used in the GOAT manuscript from the GOAT GitHub page. This file is cached in the output directory and only needs to be downloaded once. Multiple datasets are included and their names include the respective PubMed identifiers (PMID).

If you encounter technical difficulties, try to;

1. download the file by copy/pasting this URL into your browser: https://github.com/ftwkoopmans/goat/raw/main/analyses
2. load the data in R using the following 2 lines of code, here assuming you stored the downloaded file at C:/data/goat_manuscript_datasets.rda

```
load("C:/data/goat_manuscript_datasets.rda")
genelist = goat_manuscript_datasets.rda[["Wingo 2020:mass-spec:PMID32424284"]]
```

Usage

```
download_goat_manuscript_data(output_dir, ignore_cache = FALSE)
```

Arguments

output_dir	full path to the directory where the downloaded files should be stored. Directory is created if it does not exist. e.g. output_dir="~/data" on unix systems, output_dir="C:/data" on Windows, or set to output_dir=getwd() to write output to the current working directory
ignore_cache	boolean, set to TRUE to force re-download and ignore cached data, if any. Default: FALSE

Value

a list of genelist data tables. The names of the list represent the datasets, values in the list are data tables that can be used as a "genelist" in the GOAT R package

---

filter_genesets *filter a geneset table; intersect with an array of genes-of-interest then apply cutoffs on min/max genes per geneset*

Description

filter a geneset table; intersect with an array of genes-of-interest then apply cutoffs on min/max genes per geneset

Usage

```
filter_genesets(
  genesets,
  genelist,
  min_overlap = 10L,
  max_overlap = 1500L,
  max_overlap_fraction = 0.5,
  min_signif = NA,
  max_size = NA,
  dedupe = FALSE
)
```

Arguments

genesets	tibble with genesets, must contain columns 'id', 'genes' and 'ngenes'
genelist	tibble with genes, must contain column 'gene' and 'signif'. gene = character column, which are matched against list column 'genes' in genesets tibble. signif = boolean column (you can set all to FALSE if not performing Fisher-exact or hypergeometric test downstream)
min_overlap	integer, minimum number of genes in the genelist table that must match a geneset. Must be at least 1 but when using the GOAT algorithm downstream, this should be set to at least 10 (default=10). e.g. when set to 10, this will only retain genesets that contain at least 10 genes that are also in your genelist.

max_overlap integer, maximum number of genes in the genelist table that must match a geneset. Set to NA to disable

max_overlap_fraction analogous to max_overlap, which limits the max geneset size to a given N, this parameter defines the maximum geneset size that is to be retained as a fraction of the input genelist length. For example, setting this to 0.5 will remove all genesets that contain more than half the genes in the input genelist (i.e. testing enrichment of a geneset that contains 1000 out of a total 1200 genes from your input genelist is probably meaningless). Defaults to 50%

min_signif expert setting for debugging and algorithm evaluation/benchmarking, NOT for regular geneset analyses. integer, minimum number of genes in the genelist table that are signif==TRUE and match a geneset. Be careful, this is "prefiltering" and will affect the correctness / calibration of estimated geneset p-values. For GOAT and GSEA, this is NOT RECOMMENDED and will cause bias in your dataset! Set to NA to disable (default)

max_size integer, maximum number of genes in the geneset (i.e. prior to intersect with user's gene list provided as genelist). Optionally, use this to remove highly generic terms. Set to NA to disable

dedupe boolean, remove duplicate genesets (as determined after intersection with genelist)

Value

the input genesets filtered for the subset of rows that match user's filter parameters

---

gg_color_hue *generate colours analogous to ggplot's default palette*

Description

https://stackoverflow.com/a/8197703

Usage

```
gg_color_hue(n)
```

Arguments

n number of colors

Value

a color code (string)

goat_logo *ASCII logo for this package*

Description

ASCII logo for this package

Usage

```
goat_logo()
```

Value

package logo as a string

goat_nulldistributions *Precomputed parameters used by the GOAT algorithm*

Description

there parameters are used by goat to efficiently perform geneset testing without bootstrapping

Usage

```
goat_nulldistributions
```

Format

`goat_nulldistributions`:
a data.frame with precomputed GOAT null distribution parameters

goat_print_version *Print package version and logo to console*

Description

Print package version and logo to console

Usage

```
goat_print_version()
```

Value

prints to console without returning a value

goat_version *Return goat package version as a string*

Description

simple wrapper around utils::packageVersion()

Usage

```
goat_version()
```

Value

package version as a string

go_gene2go *parse gene2go file*

Description

note that this file lacks parent/child relations, so we only learn 'direct annotations'

Usage

```
go_gene2go(f, taxid_filter = 9606)
```

Arguments

f	full path to gene2go file stored on the computer, e.g. previously downloaded from https://ftp.ncbi.nih.gov/gene/DATA/gene2go.gz
taxid_filter	taxonomy id, integer

Value

a tibble with columns; source, source_version, id, name, genes, ngenes

go_obo — *simple vectorized parsing of GO OBO file without any dependencies (beyond dplyr/tibble/tidyr)*

Description

note that we remove links between GO terms that are across GO domains (e.g. no CC to MF relations) The only supported relations are those that match this regex; `"^(is_a:|relationship: part_of|relationship: regulates|relationship: positively_regulates|relationship: negatively_regulates`

Usage

```
go_obo(f, rename_namespace = TRUE, remove_obsolete = TRUE)
```

Arguments

`f`	full path to go.obo file stored on the computer, e.g. previously downloaded from http://current.geneontology.org/ontology/go.obo
`rename_namespace`	boolean; rename official namespace values like 'cellular_component' to CC? (analogous for BP and MF)
`remove_obsolete`	boolean; remove obsoleted terms?

Value

tibble with ontology terms and their relations

hgnc_idmap_table — *Parse HGNC gene identifier lookup table that was downloaded from genenames.org into a table with HGNC ID, symbol, synonym (NA if unavailable), entrez ID*

Description

download link: https://www.genenames.org/download/statistics-and-files/ table: "Complete dataset download links" –» "Complete HGNC approved dataset" –» download the "TXT" table filename is typically something like hgnc_complete_set.txt URL; https://ftp.ebi.ac.uk/pub/databases/genenames/hgnc/tsv/hgnc_complete

Usage

```
hgnc_idmap_table(filename)
```

Arguments

`filename`	full path to the downloaded table (expected to be tsv format, typically has .txt or .tsv file extension)

Value

a long-format table with columns; hgnc_id, hgnc_symbol, type, value

alternatively;

table: "Total Approved Symbols" –» "TXT" / "text file in TSV format" filename is typically something like non_alt_loci_set.txt

---

lighten_color *naively lighten a color by mixing in white*

Description

naively lighten a color by mixing in white

Usage

```
lighten_color(color, frac = 0.1)
```

Arguments

color	input colors
frac	fraction of white; >0 and <1

Value

adjusted value for input color

---

load_genesets_gmtfile *parse genesets in GMT format where gene identifiers are numeric Entrez gene IDs*

Description

parse genesets in GMT format where gene identifiers are numeric Entrez gene IDs

Usage

```
load_genesets_gmtfile(filename, label)
```

Arguments

filename	input file for this function should be the full path to genesets defined in GMT format
label	a shortname for the genesets in this file, for example "GO_CC", "KEGG", "MY_DB_V1". This will be stored in the 'source' column of the resulting table. Importantly, multiple testing correction in GOAT is grouped by this 'source' column so you probably want to use a different label for each collection-of-genesets that you load. Must not be empty, only allowed characters are; upper/lower-case letter, numbers 0-9 and underscore

Value

tibble with columns; source (character), source_version (character), id (character), name (character), genes (list), ngenes (int)

Example data;

URL: https://www.gsea-msigdb.org/gsea/msigdb/human/collections.jsp#C5 download this data: KEGG subset of curated pathways –» NCBI (Entrez) Gene IDs filename should be something like "c2.cp.kegg.v2023.1.Hs.entrez.gmt

Examples

```
# TODO: update the filename to your downloaded file
f = "C:/DATA/c2.cp.kegg.v2023.1.Hs.entrez.gmt"
if(file.exists(f)) genesets_asis = load_genesets_gmtfile(f, label = "KEGG")
```

---

load_genesets_go_bioconductor

human gene (NCBI entrez ID) annotations from the GO database using the 'org.Hs.eg.db' Bioconductor package

---

Description

Download and import genesets from the GO database using the Bioconductor infrastructure. Use the `goat::load_genesets_go_fromfile` function for more fine-grained control over the GO database version that you use; it allows you to import NCBI gene2go files

Usage

```
load_genesets_go_bioconductor(include_child_annotations = TRUE)
```

Arguments

include_child_annotations	boolean; include annotations against child terms? In most situations, TRUE (default) is the desired setting

Details

Note that org.Hs.eg.db pulls data semi-annually from NCBI gene2go, but the GO database version returned by this function is tied to the version of the org.Hs.eg.db on your computer (this is controlled by the Bioconductor infrastructure).

The actual GO database version that is retrieved is returned by this function in the `source_version` column.

Value

table with columns; source (character), source_version (character), id (character), name (character), genes (list), ngenes (int)

`load_genesets_go_fromfile` *construct a geneset table from gene2go and OBO files*

Description

This function is used to load Gene Ontology (GO) genesets from files that you manually downloaded from the links below. This enables the use of the latest data from GO (in contrast, Bioconductor GO data may lag behind current data considerably). To construct genesets from available raw data, download the "gene2go" file (the gene annotations) from below NCBI link and download the GO OBO (ontology terms and relations to respective parent/child terms) from below geneontology.org link. Provide the full path to the downloaded file to this function. Both "gzipped" and "uncompressed" files are supported.

We encourage you to rename the files after your downloaded them such that the date of download in incorporated; this ensures you can always keep track of the GO database version that was used! For example, rename the downloaded "gene2go.gz" file to "gene2go_2024-01-31.gz".

Download link for gene2go file; https://ftp.ncbi.nih.gov/gene/DATA/gene2go.gz

Download link for gene ontology OBO file; http://current.geneontology.org/ontology/go.obo

Usage

```
load_genesets_go_fromfile(
  file_gene2go,
  file_goobo,
  include_child_annotations = TRUE
)
```

Arguments

`file_gene2go`	full path to the gene2go file from NCBI. Also works with the gzipped file gene2go.gz
`file_goobo`	full path to the OBO file from geneontology.org
`include_child_annotations`	boolean; include annotations against child terms? In most situations, TRUE (default) is the desired setting

Value

table with columns; source (character), source_version (character), id (character), name (character), genes (list), ngenes (int)

Examples

```
# TODO: update the filenames to your downloaded files
file_gene2go = "C:/DATA/gene2go_2024-01-01.gz"
file_goobo = "C:/DATA/go_2024-01-01.obo"
if(file.exists(file_gene2go) && file.exists(file_goobo)) {
  genesets_asis = load_genesets_go_fromfile(file_gene2go, file_goobo)
}
```

---

load_genesets_syngo *parse genesets from the SynGO database*

Description

Workflow;

- obtain the input file from; https://www.syngoportal.org
- click "bulk download SynGO release ..." for SynGO release of interest
- unzip
- call this function with the full file path to the 'syngo_ontologies.xlsx' file

Usage

```
load_genesets_syngo(filename, gene_database = "entrez")
```

Arguments

filename	full path to the "syngo_ontologies.xlsx" file that was extracted from a SynGO bulk download ZIP archive
gene_database	gene IDs to return. must be any of; "entrez" (default), "hgnc", "ensembl"

Value

table with columns; source (character), source_version (character), id (character), name (character), genes (list), ngenes (int)

Examples

```
# TODO: update the filename to your downloaded file
f = "C:/DATA/SynGO_bulk_download_release_20231201/syngo_ontologies.xlsx"
if(file.exists(f)) genesets_asis = load_genesets_syngo(f)
```

`minlog10_fixzero` *-log10 transform a vector of p-values, replacing zeros with some limit/threshold*

Description

-log10 transform a vector of p-values, replacing zeros with some limit/threshold

Usage

```
minlog10_fixzero(x, limit = 2.22e-16)
```

Arguments

`x`	p-value vector to transform to -log10
`limit`	value to replace zero's in `x` with. Set NA to replace zero's in `x` with the smallest finite value in `x` (if there is none, defaults to 2.22e-16)

Value

input parameter `x` transformed to -log10

Examples

```
pval = c(0, 10^-6, 0.001, 0.01, 1, NA, -Inf, Inf, NaN)
cbind(
  input = pval,
  # default; replace zeros with typical R machine precision for doubles
  minlog10_default = minlog10_fixzero(pval),
  # alternatively, replace zero with lowest non-zero pvalue in input
  minlog10_limit_from_data = minlog10_fixzero(pval, limit = NA)
)
```

`padjust_genesets` *Adjust p-values for all genesets, grouped by 'source' then adjust for the number of 'sources'*

Description

Adjust p-values for all genesets, grouped by 'source' then adjust for the number of 'sources'

Usage

```
padjust_genesets(
  genesets,
  method = "BH",
  cutoff = 0.01,
  correct_sources = TRUE
)
```

Arguments

genesets	tibble with genesets, must contain column 'pvalue'
method	method for multiple testing correction, must be any of stats::p.adjust.methods, e.g. "BH" or "bonferroni"
cutoff	numeric cutoff value for adjusted p-value, signif column is set to TRUE for all values lesser-equals
correct_sources	apply Bonferroni adjustment to all p-values according to the number of geneset sources that were tested. Boolean parameter, set TRUE to enable (default) or FALSE to disable

Value

updated genesets table

---

partition_genes *Classify genes into 2 groups, e.g. to define significant or topN genes, resulting in a 'signif' column with boolean values*

Description

This can be convenient to prepare the significant/test/foreground set for classical ORA, e.g. test_genesets() with parameter method = "fisherexact". Note that the GOAT geneset enrichment algorithm does not use data in the 'signif' column of the input genelist.

Usage

```
partition_genes(
  genes,
  col,
  decreasing = FALSE,
  use_abs = FALSE,
  cutoff = NULL,
  fraction = NULL,
  topn = NULL
)
```

Arguments

genes	gene tibble where each row is a unique gene, must contain column name col
col	column name in genes
decreasing	order col in descending (set TRUE) or ascending order (set FALSE, default) prior to partitioning?
use_abs	use absolute values (default FALSE), e.g. when setting a threshold on effect-sizes

cutoff	threshold for values in col to select (must provide exactly 1 parameter for filtering, either cutoff, fraction or topn)
fraction	fraction of rows in genes tibble to select (must provide exactly 1 parameter for filtering, either cutoff, fraction or topn)
topn	number of rows in genes tibble to select (must provide exactly 1 parameter for filtering, either cutoff, fraction or topn)

Value

input table genes with results in the "signif" column

Examples

```
# note: this example will download 1 files of approx 4MB

# store the downloaded files in the following directory. Here, the temporary file
# directory is used. Alternatively, consider storing this data in a more permanent location.
# e.g. output_dir="~/data/goat" on unix systems or output_dir="C:/data/goat" on Windows
output_dir = tempdir()

# Download an example gene list, i.e. one of the datasets analyzed in the GOAT manuscript.
datasets = download_goat_manuscript_data(output_dir)
genelist = datasets$`Wingo 2020:mass-spec:PMID32424284`

# example 1: significant hits
genelist = partition_genes(genelist, col="pvalue_adjust", decreasing=FALSE, cutoff=0.01)
cat(sum(genelist$signif), "/", nrow(genelist), "are signif\n")

# example 2: abs(effectsize) >= 5
genelist = partition_genes(genelist, col="effectsize", decreasing=TRUE, use_abs=TRUE, cutoff=5)
cat(sum(genelist$signif), "/", nrow(genelist), "are signif\n")

# example 3: top 10% 'best' p-values
genelist = partition_genes(genelist, col="pvalue", decreasing=FALSE, fraction = 0.1)
cat(sum(genelist$signif), "/", nrow(genelist), "are signif\n")
```

---

plot_heatmap *plot the geneset similarity matrix as a heatmap*

Description

plot the geneset similarity matrix as a heatmap

Usage

```
plot_heatmap(
  x,
  output_dir,
  colors = grDevices::hcl.colors(100, "Viridis", rev = FALSE),
  fontsize = 10
)
```

Arguments

x	result from cluster_genesets()
output_dir	set to NA to directly show the figures instead of writing them to file. Otherwise, this is the full path to the directory where the downloaded files should be stored. Directory is created if it does not exist. e.g. output_dir="~/data" on unix systems, output_dir="C:/data" on Windows, or set to output_dir=getwd() to write output to the current working directory
colors	a vector of 100 colors to be used for the heatmap (101 breaks are computed between 0 and the max value in the distance matrix)
fontsize	parameter sent to pheatmap::pheatmap(); control the size of labels in the plot, defaults to 10. Note that you can also change the plot device size, see examples

Value

does not return a value, plots are printed to device or files depending on output_dir parameter

Examples

```
# note; this example downloads data when first run, and typically takes ~60seconds

# store the downloaded files in the following directory. Here, the temporary file
# directory is used. Alternatively, consider storing this data in a more permanent location.
# e.g. output_dir="~/data/goat" on unix systems or output_dir="C:/data/goat" on Windows
output_dir = tempdir()

## first run the default example from test_genesets() to obtain geneset results
datasets = download_goat_manuscript_data(output_dir)
genelist = datasets$`Wingo 2020:mass-spec:PMID32424284`
genesets_asis = download_genesets_goatrepo(output_dir)
genesets_filtered = filter_genesets(genesets_asis, genelist)
result = test_genesets(genesets_filtered, genelist, method = "goat",
  score_type = "effectsize", padj_method = "bonferroni", padj_cutoff = 0.05)

# prior to running this function, cluster the genesets
clusters = cluster_genesets(result, genelist)

# use the plot heatmap function and try various color palettes
plot_heatmap(clusters, output_dir, colors = hcl.colors(100, "Viridis", rev = FALSE))
plot_heatmap(clusters, output_dir, colors = hcl.colors(100, "Inferno", rev = FALSE))
plot_heatmap(clusters, output_dir, colors = hcl.colors(100, "Lajolla", rev = TRUE))
plot_heatmap(clusters, output_dir, colors = hcl.colors(100, "Mako", rev = FALSE))
```

```
plot_heatmap(clusters, output_dir, colors = hcl.colors(100, "Turku", rev = TRUE))
plot_heatmap(clusters, output_dir, colors = hcl.colors(100, "Grays", rev = TRUE))
```

`plot_lollipop` *Lollipop chart or barplot visualization of geneset enrichment testing results*

Description

Lollipop chart or barplot visualization of geneset enrichment testing results

Usage

```
plot_lollipop(
  x,
  output_dir,
  only_reduced = FALSE,
  plot_type = "lollipop",
  show_pvalue = FALSE,
  score_xaxis = "minlogp",
  score_color = ifelse(is.data.frame(x) && "score_type" %in% colnames(x) &&
    is.character(x$score_type) && any(x$score_type %in% c("effectsize_up",
    "effectsize_down")), "updown", "minlogp"),
  score_color_limits = "source",
  score_color_updown = c("#E57373", "#5E7ABC"),
  max_ngenes = NA,
  topn = NA,
  padj_cutoff = NA
)
```

Arguments

`x`	results from function `test_genesets`
`output_dir`	set to NA to directly show the figures instead of writing them to file. Otherwise, this is the full path to the directory where the downloaded files should be stored. Directory is created if it does not exist. e.g. `output_dir="~/data"` on unix systems, `output_dir="C:/data"` on Windows, or set to `output_dir=getwd()` to write output to the current working directory
`only_reduced`	only show the reduced/summarized set of significant genesets. This requires that you first applied the `reduce_genesets` function
`plot_type`	Options: "barplot", "lollipop" (default)
`show_pvalue`	boolean parameter that indicates whether adjusted p-values should be shown next to each data point

score_xaxis
: type of score to show on the x-axis. Options: "minlogp" to show -log10(adjusted pvalue), which is default. Use "oddsratio" to show the enrichment of significant genes. For further details on this score and its computation, see the score_geneset_oddsratio function documentation. Basically, the genesets in this plot are sorted by their proportion of foreground/significant genes (and this ratio is standardized against the overall ratio of significant genes as to make this statistic comparable across analyses/datasets).

score_color
: analogous to score_xaxis, here you can specify the data used for color-coding the plot. Options: "minlogp", "oddsratio", "updown". The former 2 options are the same as for score_xaxis, the latter enables color-coding by up-/down-regulation as encoded in the "score_type" column. Note that this only works when geneset testing based on "effectsize" was performed and thus the "score_type" column has values "effectsize_up" or "effectsize_down" (encoding directionality). Rows with other values are assumed NA and colored as grey.

score_color_limits
: defines the limits for the color scales. options; score_color_limits = "source" use the range of values per 'source' to compute colors scales (default). Set to "overall" in order to have a unified color scale across 'source' (e.g. same color bar across GO_CC/GO_MF/GO_BP). Alternatively, provide a numeric vector of 2 values to manually define lower/upper-limits.

score_color_updown
: array of 2 strings that describe the colors for up- and down-regulation (used when score_color is set to "updown"). Default color-coding; up = red, down = blue. Use hex color codes, e.g. "#ff0000" for red.

max_ngenes
: only plot terms with less than N genes (quick way to get rid of large/unspecific terms)

topn
: topn terms to show after sorting genesets by p-value. For example, this makes it easy to plot the top10 GO terms. Set to NA to ignore this filter (default)

padj_cutoff
: adjusted pvalue cutoff for terms shown in the plot. If set to NA (default), all significant genesets are used (i.e. 'signif' column in the input geneset table)

Value

if output_dir is NA, a list of ggplot2 objects. Otherwise, write plots to file and do not return any value

Examples

```
# note; this example downloads data when first run, and typically takes ~60seconds

# store the downloaded files in the following directory. Here, the temporary file
# directory is used. Alternatively, consider storing this data in a more permanent location.
# e.g. output_dir="~/data/go" on unix systems or output_dir="C:/data/go" on Windows
output_dir = tempdir()

## first run the default example from test_genesets() to obtain geneset results
datasets = download_goat_manuscript_data(output_dir)
genelist = datasets$`Wingo 2020:mass-spec:PMID32424284`
```

```
genesets_asis = download_genesets_goatrepo(output_dir)
genesets_filtered = filter_genesets(genesets_asis, genelist)
result = test_genesets(genesets_filtered, genelist, method = "goat",
  score_type = "effectsize", padj_method = "bonferroni", padj_cutoff = 0.05)

# generate lollipop charts for each GO domain (CC/BP/MF), with geneset -log10
# adjusted p-value on the x-axis and color-coding by geneset up/down-regulation
plot_lollipop(
  result, output_dir, plot_type = "lollipop", topn = 50,
  score_xaxis = "minlogp", score_color = "updown"
)

# alternatively, as a barplot
plot_lollipop(
  result, output_dir, plot_type = "barplot", topn = 50,
  score_xaxis = "minlogp", score_color = "updown"
)

# alternatively, color-code genesets by enrichment of significant genes using
# parameter `score_color="oddsratio"` . See further `score_geneset_oddsratio`
# function documentation for definition/computation of this score.
plot_lollipop(
  result, output_dir, plot_type = "lollipop", topn = 50,
  score_xaxis = "minlogp", score_color = "oddsratio"
)
```

`plot_network` *plot geneset distance matrix as a network*

Description

plot geneset distance matrix as a network

Usage

```
plot_network(
  clusters,
  src,
  show_clusters = TRUE,
  show_text = FALSE,
  topn_edges = 5,
  clr_default = "#29b6f6",
  node_color_palette = goat::gg_color_hue
)
```

Arguments

`clusters`	result from `cluster_genesets`

src	source property (e.g. "GO_CC")
show_clusters	boolean value
show_text	boolean value
topn_edges	topN edges to retain per geneset (typically 5~8)
clr_default	default color for the network, used only when show_clusters is set to FALSE
node_color_palette	function with 1 parameter, N, that returns N colors (default is goat function gg_color_hue)

Value

a ggplot2 object

`plot_volcano` *For each provided geneset, a volcano plot of all genelist log2fc and p-values with respective geneset constituents highlighted*

Description

For each provided geneset, a volcano plot of all genelist log2fc and p-values with respective geneset constituents highlighted

Usage

```
plot_volcano(
  x,
  genelist,
  plot = TRUE,
  topn_labels = 0,
  color_default = "#B0B0B0",
  color_highlight = "#ef5350",
  color_label = "#000000",
  pointsize = 2,
  pointalpha = 0.75,
  labelsize = 7
)
```

Arguments

x	a subset of the results from test_genesets function, see example
genelist	input genelist, must contain columns 'gene', 'log2fc' and 'pvalue_adjust' (not! log transformed). If parameter topn_labels is provided, also include a character column 'symbol' that contains gene names/symbols/labels
plot	if TRUE, will directly show the plots. if FALSE, returns a list of ggplot objects corresponding to rows in the input result parameter

topn_labels	for how many genes that overlap between genelist and a geneset should we plot the gene symbol? This requires a column 'symbol' in the genelist parameter (default: 0)
color_default	color for genes that are not part of a geneset (default: grey)
color_highlight	color used to highlight geneset constituents (default: red)
color_label	provided that topn_labels is set, this is the color of the text labels (default: black)
pointsize	size of the dots, this parameter is passed to geom_point (default: 2)
pointalpha	alpha of the dots, this parameter is passed to geom_point (default: 0.75)
labelsize	provided that topn_labels is set, this is the text size (in pt) for the labels (default: 7)

Value

if `plot==FALSE`, a list of ggplot2 objects. Otherwise, does not return any value

Examples

```
# note; this example downloads data when first run, and typically takes ~60seconds

# store the downloaded files in the following directory. Here, the temporary file
# directory is used. Alternatively, consider storing this data in a more permanent location.
# e.g. output_dir="~/data/goat" on unix systems or output_dir="C:/data/goat" on Windows
output_dir = tempdir()

## first run the default example from test_genesets() to obtain geneset results
datasets = download_goat_manuscript_data(output_dir)
genelist = datasets$`Wingo 2020:mass-spec:PMID32424284`
genesets_asis = download_genesets_goatrepo(output_dir)
genesets_filtered = filter_genesets(genesets_asis, genelist)
result = test_genesets(genesets_filtered, genelist, method = "goat",
  score_type = "effectsize", padj_method = "bonferroni", padj_cutoff = 0.05)

## example 1; select top10 GO CC terms from the geneset testing results
result_subset = result |> filter(source == "GO_CC") |> arrange(pvalue) |> head(n = 10)
pdf(paste0(output_dir, "/volcano_CC_top10.pdf"), width = 4, height = 4)
plot_volcano(result_subset, genelist)
dev.off()

## example 2;, select small genesets that are significant and have
## near-exclusive enrichment in either up up/down-regulated genes
# first, add geneset directionality scores to our results
result = score_geneset_directionality(result, genelist)
# next, subset the geneset result table
result_subset = result |>
  filter(signif & ngenes <= 50 & abs(score_directionality_rank) > 0.6) |>
  arrange(pvalue_adjust)
# finally, create plots. Note that the genelist contains a column 'symbol'
# which we use here to print labels for the topN genes per plotted geneset
pdf(paste0(output_dir, "/volcano_signif_ngenes50_directionality06.pdf"), width = 4, height = 4)
```

```
plot_volcano(result_subset, genelist, topn_labels = 10)
dev.off()
```

`rankscore` *compute rank^2 scores and rescale these between 0~1000 (with further precision captured by decimals)*

Description

rank^2 can yield huge values, e.g. a large genelist of N=50000 genes would imply max gene score = 50000^2 = 2.5e+09. Thus we rescale these scores between 0~1000 so downstream applications (e.g. sum of 10000 gene scores) don't explode to huge numbers.

Usage

```
rankscore(x, sort1, sort2, sort3, colname)
```

Arguments

`x`	data.frame to sort (i.e. the genelist)
`sort1`	numeric vector of length `nrow(x)` to sort by first/primarily (descending order, higher value gets better result score)
`sort2`	numeric vector of length `nrow(x)` to sort by for breaking ties (descending order, higher value gets better result score)
`sort3`	numeric vector of length `nrow(x)` to sort by for breaking ties (descending order, higher value gets better result score)
`colname`	name for result column in `x` (overwritten if already exists)

Value

data.frame `x` with added column `colname`, containing gene scores between 0 and 1000

Examples

```
x = data.frame(gene=c(1, 2, 3, 4, 5),
               pvalue=c(0.01, 1, 1, 0.1, 1),
               effectsize=c(-2, 0.25, 0.5, 1, 0.25))
print(x, row.names = FALSE)
print(rankscore(x, sort1 = -1*x$pvalue, sort2 = abs(x$effectsize),
      sort3 = x$gene, colname="score") |>
  arrange(desc(score)), row.names = FALSE)
```

rankscore_fixed_order *Gene score array, from low to high scores*

Description

Gene score array, from low to high scores

Usage

```
rankscore_fixed_order(n)
```

Arguments

n	genelist length

Value

result of `(1:n)^2 / (n^2 / 1000)`

reduce_genesets *Reduce the set of significant genesets to a minimum*

Description

Analyses are performed independently per 'source' of genesets. The result of this function is the geneset table with a newly appended column 'signif_and_reduced'

Usage

```
reduce_genesets(
  clusters,
  simscore_threshold = 0.9,
  universe_fraction = 0.25,
  signifgenes_fraction = 0.9
)
```

Arguments

clusters	results from `cluster_genesets()`
simscore_threshold	similarity score (0~1) that is required to consider one geneset to be a "parent term" of another. Setting a lower value will yield fewer genesets / stronger summarization. Typical settings for this parameter are 0.8~0.99 (0.9 is default)

`universe_fraction`
: discard genesets that cover more than X fraction of all genes in the universe (unique set of genes covered by all significant genesets). Setting this to 0.25 will deprioritize genesets that cover 25% of all genes (in significant genesets). This prevents very generic GO terms like "protein-containing complex" to be included in results. Typical settings for this parameter are 0.1~0.5 (0.25 is default)

`signifgenes_fraction`
: the minimum fraction of "foreground genes" ('genes_signif' column) found across all significant genesets that should be covered by the reduced geneset collection. This parameter doesn't do anything if there are fewer than 5 "foreground genes" alltogether. Typical settings for this parameter are 0.75~0.95 (0.9 is default)

Value

the genesets table from the `clusters` parameter, with results in column "signif_and_reduced"

`save_genesets` *Write a geneset table to file.*

Description

Works for any filtered geneset table generated by this package, e.g. results from any of these functions; `filter_genesets()`, `test_genesets()` or `simplify_genesets()`.

The genelist table is required to 1) lookup gene symbols and 2) determine the ordering of genes within-geneset. The latter makes it such that the output table shows the most important genes (in context of user's data) first.

All output columns that list gene identifiers or symbols are trimmed to 25000 characters. Depending on your input gene identifier type and dataset filtering settings (e.g. max geneset size), this might lead to some truncation. For example, 15 characters per ensembl gene ID (if these are used in provided genesets instead of default NCBI Entrez) and 1 char as delimiter, only the top ~1500 ensembl gene identifiers are included. If your geneset filtering allows for larger genesets and you want to check if any gene identifiers are lost upon saving, keep an eye out for the trailing ellipsis ('...'). Because genes in each geneset of the output table are sorted by order of importance as indicated in the genelist table, this shouldn't be an issue in typical use-cases where the output table is to check the topN most important/significant genes of a geneset of interest.

Usage

```
save_genesets(x, genelist, filename, arrange_genes = TRUE)
```

Arguments

`x`
: result from `filter_genesets()`, `test_genesets()` or `simplify_genesets()`. When saving the latter, output will include clusters assigned to each significant geneset

genelist	same as provided for `test_genesets()`. A column named 'symbol' is required, it'll be used to pretty-print gene symbols per geneset in the output table
filename	full path to the output file. Supported file extensions; csv, tsv, xlsx. Optionally, set to NA to not write to disk and return the result table instead
arrange_genes	set to TRUE (default) to arrange the genelist table by best p-value on top (if column 'pvalue' exists), alternatively by descending absolute effectsize (if no pvalue but effectize is available). Set FALSE to use sorting of the genelist table as-is

Value

if `filename` is `NA`, returns the validated and formatted geneset result table. Otherwise, writes the table to file and does not return a value

---

`score_geneset_directionality`

Compute a score between -1 and 1 representing the proportion of up- or down-regulated genes for each geneset, weighted by gene effectsizes

---

Description

Importantly, the scope/utility of this score is limited to help users post-hoc filter for genesets that contain mostly up/down-regulated genes. However, this might not coincide with the geneset pvalues / significance. For example, genesets may exclusively contain genes with a positive effectsize but at the same time these can all be minor effects/values and thus the geneset is not significant or less significant than other genesets with the exact same "directionality score". For example, genesets may contain both up- and down-regulated genes but still be significant when testing with GOAT and using `score_type='effectsize'`

The scores computed with this function can help in post-hoc interpretation of GOAT results to further narrow down all significant genesets to a subset with strong directionality. For example, after `test_genesets()` we can filter the results for A) significant genesets and B) that contain at most N genes and C) that are near-exclusively up/down-regulated. Bringing this all together (also useful for other types of geneset testing, like ORA, score_type="pvalue", etc); `result = test_genesets(genesets_filtered, genelist, method = "goat", score_type = "effectsize", padj_method = "bonferroni", padj_cutoff = 0.05) result = score_geneset_directionality(result, genelist) result |> filter(signif & ngenes <= 100 & abs(score_directionality_rank) >= 0.95)`

Usage

```
score_geneset_directionality(genesets, genelist)
```

Arguments

genesets	tibble with genesets, must contain columns 'source', 'id', 'genes'
genelist	tibble with genes, must contain columns 'gene', 'effectsize'

Value

input `genesets` with results in 3 columns; `score_directionality_rank` is the weighted gene score where gene values are the sign of their effectsize and weights are linearly proportional to their inverse ranks in the input genelist. `score_directionality_rank2` is similar, but now using rank^2 gene weights to boost the influence of most-important genes in the input genelist. `score_directionality_value` uses the absolute gene effectsizes as gene weights Note that the latter is least robust as it depends on the scaling of input data!

score definitions;

geneset directionality score = weighted mean of respective genes, where gene weights are 1 minus their relative rank in up/down-regulation (depending on negative/positive effectsize) and the value for each gene is -1 or 1 depending on up/down-regulation (sign of effectsize).

pseudocode;

1. gene values and weights A) gene weight between 0 and 1 for the subset of upregulated genes / positive effectsizes;

- r = for the subset of genes with effectsize > 0, compute gene rank (1 = highest effectsize, N = smallest effectsize that is greater than zero)
- weight = 1 - r/max(r) B) analogous to (A) for the subset of genes with negative effectsize C) result per gene: value = sign of its effectsize, weight = 0 if effectsize 0, otherwise respective weights from (A) or (B)

1. geneset score_directionality = weighted mean over values/weights of respective genes

---

`score_geneset_oddsratio`

Compute odds-ratio for each geneset

Description

gs_signif = number of significant genes in geneset G that intersect with user's genelist (i.e. foreground genes in G) gs_all = number of genes in geneset G that intersect with user's genelist (i.e. foreground+background genes in G) k_signif = total number of *significant* genes in user's genelist k_all = total number of genes in user's genelist

gs_signif/gs_all = ratio of foreground genes in geneset G k_signif/k_all = ratio of overall foreground genes (i.e. expected value for a random geneset)

oddsratio = (gs_signif/gs_all) / (k_signif/k_all)

Usage

```
score_geneset_oddsratio(genesets, genelist)
```

Arguments

genesets	tibble with genesets, must contain columns 'source', 'id', 'ngenes', 'ngenes_signif'
genelist	tibble with genes, must contain columns 'gene', 'signif'

Value

input genesets with results in column "score_oddsratio"

---

string_trunc_right *simple string truncation*

Description

replacement for stringr::trunc() so we don't need a package dependency for just 1 function (our code was adapter therefrom)

Usage

```
string_trunc_right(string, width, trim_left = FALSE)
```

Arguments

string	string that should be truncated
width	desired max length
trim_left	instead of right trunc (default), do left instead

Value

truncated variant of input string

---

symbol_to_entrez *Map the the symbol column in a table to HGNC human gene IDs by matching official gene symbols and synonyms*

Description

Map the the symbol column in a table to HGNC human gene IDs by matching official gene symbols and synonyms

Usage

```
symbol_to_entrez(x, hgnc)
```

Arguments

x	a data.table with a column symbol
hgnc	HGNC lookup table from `hgnc_idmap_table()`

Value

entrez gene IDs are returned in the "gene" column of table `x`. Additionally, columns "entrez_id", "hgnc_id" and "hgnc_symbol"

Examples

```
# TODO: update the filename to your downloaded file
# download instructions in the documentation of `hgnc_idmap_table()`
f = "C:/DATA/HGNC/hgnc_complete_set.txt"

if(file.exists(f)) {
  df = data.frame(symbol = c("vamp2", "STXBP1", "UNC18", NA, "PSD95", "NOT-A-GENE"))
  hgnc = hgnc_idmap_table(f)
  df = symbol_to_entrez(df, hgnc)
  print(df)
}
```

---

test_genesets *Perform geneset enrichment testing using any supported method*

Description

Perform geneset enrichment testing using any supported method

Usage

```
test_genesets(
  genesets,
  genelist,
  method,
  padj_method = "BH",
  padj_sources = TRUE,
  padj_cutoff = 0.01,
  padj_min_signifgenes = 0L,
  ...
)
```

Arguments

genesets	tibble with genesets, must contain columns 'source', 'source_version', 'id', 'name', 'genes', 'ngenes', 'ngenes_signif'

genelist	tibble with genes, must contain column 'gene' and 'test'. gene = character column, which are matched against list column 'genes' in genesets tibble. test = boolean column (you can set all to FALSE if not performing Fisher-exact or hypergeometric test downstream)
method	method for overrepresentation analysis. Options: "goat", "hypergeometric", "fisherexact", "fisherexact_ease", "gsea", "idea"
padj_method	first step of multiple testing correction; method for p-value adjustment, passed to stats::p.adjust() via padjust_genesets(), e.g. set "BH" to compute FDR adjusted p-values (default) or "bonferroni" for a more stringent procedure
padj_sources	second step of multiple testing correction; apply Bonferroni adjustment to all p-values according to the number of geneset sources that were tested. Boolean parameter, set TRUE to enable (default) or FALSE to disable
padj_cutoff	cutoff for adjusted p-value, signif column is set to TRUE for all values lesser-equals
padj_min_signifgenes	if a value larger than zero is provided, this will perform additional post-hoc filtering; after p-value adjustment, set the pvalue_adjust to NA and signif to FALSE for all genesets with fewer than padj_min_signifgenes 'input genes that were significant' (ngenes_signif column in genesets table). So this does not affect the accuracy of estimated p-values, in contrast to prefiltering genesets prior to p-value computation or adjusting p-values
...	further parameters are passed to the respective stats method

Details

After application of the enrichment testing algorithm (e.g. GOAT, ORA or GSEA), multiple testing correction is applied to obtain adjusted p-values using padjust_genesets. That function will first apply the specified pvalue adjustment procedure in the padj_method parameter within each 'source' in the genesets table. Second, it applies Bonferroni adjustment to all p-values according to the number of different geneset sources that were tested (or set padj_sources = FALSE to disable).

For example, if the input is a genesets table that contains GO_CC, GO_BP and GO_MF genesets, first multiple testing correction is applied within each source (e.g. using FDR if so desired) and afterwards a Bonferroni correction is applied based on 3 repeated analyses.

Note that this is more rigorous than typical GO tools; hypothetically, one could split all GO_CC pathways into 1000 different databases/'sources' and then run enrichment testing. Consequently, the multiple testing burden is reduced if one doesn't adjust p-values for the number of 'sources' as we do here.

Value

the input genesets, with results stored in columns 'pvalue', 'pvalue_adjust' and 'signif'

Examples

```
#' # note; this example downloads data when first run, and typically takes ~60seconds

## Basic example for a complete GOAT workflow
```

```
# Downloads test data to your computer and stores it at current working directory
# Refer to the GitHub documentation for elaborate documentation and a worked example

# store the downloaded files in the following directory. Here, the temporary file
# directory is used. Alternatively, consider storing this data in a more permanent location.
# e.g. output_dir="~/data/go" on unix systems or output_dir="C:/data/go" on Windows
output_dir = tempdir()

# download an example gene list
datasets = download_goat_manuscript_data(output_dir)
genelist = datasets$`Wingo 2020:mass-spec:PMID32424284`

# download GO genesets
genesets_asis = download_genesets_goatrepo(output_dir)

# filter genesets for sufficient overlap with the genelist, then apply GOAT
genesets_filtered = filter_genesets(genesets_asis, genelist)
result = test_genesets(genesets_filtered, genelist, method = "goat",
  score_type = "effectsize", padj_method = "bonferroni", padj_cutoff = 0.05)

# print first 10 rows of the result table
print(result |> select(source, name, ngenes, pvalue_adjust) |> utils::head(n=10))
```

`test_genesets_fisherexact`

Geneset ORA using Fisher-exact test

Description

In most cases, it's more convenient to call the more generic `test_genesets` function which also deals with multiple-testing correction (per geneset source)

It is assumed that the `genesets` and `genelist` parameters are in sync, i.e. `genesets` provided here is the result of the `filter_genesets()` function (using the same `genelist` table)

Same as hypergeometric for non-EASE, but slower because stats::fisher.test isn't vectorized

Only genesets with at least 1 significant gene are subjected to statistical testing (e.g. NA is returned for genesets without significant genes)

Usage

```
test_genesets_fisherexact(
  genesets,
  genelist,
  require_nsignif = 1L,
  use_ease = FALSE
)
```

Arguments

genesets	tibble with genesets, must contain columns 'id', 'ngenes' and 'ngenes_signif'
genelist	tibble with genes, must contain column 'signif'. The number of rows in this table (where signif is not NA) is assumed to be the total number of tested genes, the number of rows where signif==TRUE is assumed the total number of significant genes.
require_nsignif	minimum number of 'signif genes' that overlap with a geneset; NA pvalues are returned for genesets with ngenes_signif <= require_nsignif. This function 'prefilters' genesets, so beware that this will influence downstream multiple testing correction. Default is 1
use_ease	use a more conservative score coined by DAVID online tools @ https://david.ncifcrf.gov/helps/functional_

Value

input genesets table with results in the "pvalue" column

See Also

test_genesets

Examples

```
# note; this example downloads data when first run, and typically takes ~60seconds

# store the downloaded files in the following directory. Here, the temporary file
# directory is used. Alternatively, consider storing this data in a more permanent location.
# e.g. output_dir="~/data/goat" on unix systems or output_dir="C:/data/goat" on Windows
output_dir = tempdir()

## first run the default example from test_genesets() to obtain input data
datasets = download_goat_manuscript_data(output_dir)
genelist = datasets$`Wingo 2020:mass-spec:PMID32424284`
genesets_asis = download_genesets_goatrepo(output_dir)
genesets_filtered = filter_genesets(genesets_asis, genelist)

## example: same results between Fisher-exact and hypergeometric tests
result_hg = test_genesets_hypergeometric(genesets_filtered, genelist, require_nsignif = 3L)
result_fe = test_genesets_fisherexact(genesets_filtered, genelist, require_nsignif = 3L)
all.equal(result_hg$pvalue, result_fe$pvalue)
```

---

test_genesets_goat_bootstrap

Naive GOAT variant where we estimate null parameters for each geneset size independently

---

Description

In typical use-cases, one applies `test_genesets()` instead with parameter `method="goat"` , which in turn will use `test_genesets_goat_precomputed` (and not this function).

Usage

```
test_genesets_goat_bootstrap(
  genesets,
  genelist,
  score_type,
  niter = 5e+05,
  verbose = FALSE
)
```

Arguments

`genesets`	see `test_genesets_goat_precomputed`
`genelist`	see `test_genesets_goat_precomputed`
`score_type`	see `test_genesets_goat_precomputed`
`niter`	integer number of bootstrap iterations; at least 10000, at most 5000000
`verbose`	boolean, create debug plots

Value

see `test_genesets_goat_precomputed`

`test_genesets_goat_fitfunction` *Variant of the main GOAT function* `test_genesets_goat_precomputed` *that does not use previously prepared parameters*

Description

In typical use-cases, one applies `test_genesets()` instead with parameter `method="goat"` , which in turn will use `test_genesets_goat_precomputed` (and not this function).

Usage

```
test_genesets_goat_fitfunction(
  genesets,
  genelist,
  score_type,
  niter = 5e+05,
  verbose = FALSE
)
```

Arguments

genesets	see test_genesets_goat_precomputed
genelist	see test_genesets_goat_precomputed
score_type	see test_genesets_goat_precomputed
niter	integer number of bootstrap iterations; at least 10000, at most 5000000
verbose	boolean, create debug plots

Details

Optionally, use this function to ignore precomputed/bundled null distribution estimates and perform new permutations and function fitting (e.g. because you want to test the effect of huge niter parameter, but beware of RAM requirements)

Value

see test_genesets_goat_precomputed

---

test_genesets_goat_precomputed

Test geneset enrichment with the Geneset Ordinal Association Test (GOAT) algorithm

---

Description

In most cases, it's more convenient to call the more generic test_genesets function which also applies multiple-testing correction (per geneset source) to the geneset p-values computed by this function.

This is the canonical geneset test function for GOAT that uses precomputed null distributions that are bundled with the GOAT package

Usage

```
test_genesets_goat_precomputed(genesets, genelist, score_type)
```

Arguments

genesets	genesets data.frame, must contain columns; "source", "id", "genes", "ngenes"
genelist	genelist data.frame, must contain columns "gene" and "pvalue"/"effectsize" (depending on parameter score_type)
score_type	how to compute gene scores? Option "pvalue" uses values from the pvalue column in genelist in a one-way test for enrichment; lower p-value is better Option "effectsize" uses values from the effectsize column in genelist in a two-way test for enrichment; is a geneset enriched in either down- or up-regulated genes? Option "effectsize_abs" uses values from the effectsize column in genelist in a one-way test for enrichment; is a geneset enriched when

testing absolute effectsizes? Option "effectsize_up" uses values from the effectsize column in `genelist` in a one-way test for enrichment; is a geneset enriched in up-regulated genes? (i.e. positive effectsize) Option "effectsize_down" uses values from the effectsize column in `genelist` in a one-way test for enrichment; is a geneset enriched in down-regulated genes? (i.e. negative effectsize)

Value

input `genesets` table with results in the "pvalue", "score_type" columns. "zscore" column: A standardized z-score is computed from geneset p-values + effectsize direction (up/down) if tested. Importantly, we here return standardized z-scores because the GOAT geneset score (mean of gene scores) is relative to the respective geneset-size-matched null distributions (a skewed normal)! In contrast, the standardized z-scores are comparable between genesets (as are the pvalues).

Only if either (or both) the effectsize-up/down was tested, the direction of regulation has been tested (effectsize_abs and pvalue score types are agnostic to up/down regulation). So when score_type was set to any of effectsize/effectsize_down/effectsize_up, the z-scores are negative values in case the "score_type" output column is "effectsize_down".

See Also

`test_genesets`

Examples

```
# note; this example downloads data when first run, and typically takes ~60seconds

# store the downloaded files in the following directory. Here, the temporary file
# directory is used. Alternatively, consider storing this data in a more permanent location.
# e.g. output_dir="~/data/goat" on unix systems or output_dir="C:/data/goat" on Windows
output_dir = tempdir()

## first run the default example from test_genesets() to obtain input data
datasets = download_goat_manuscript_data(output_dir)
genelist = datasets$`Wingo 2020:mass-spec:PMID32424284`
genesets_asis = download_genesets_goatrepo(output_dir)
genesets_filtered = filter_genesets(genesets_asis, genelist)

### we here compare GOAT with precomputed null distributions against
### a GOAT function that performs bootstrapping to compute null distributions on-demand

# apply goat with precomputed null (default) and goat with on-demand bootstrapping
result_precomputed = test_genesets(genesets_filtered, genelist, method = "goat",
  score_type = "effectsize", padj_method = "bonferroni", padj_cutoff = 0.05) |>
  # undo sorting by p-value @ test_genesets(), instead sort by stable IDs
  arrange(source, id)
result_bootstrapped = test_genesets(genesets_filtered, genelist, method = "goat_bootstrap",
 score_type = "effectsize", padj_method = "bonferroni", padj_cutoff = 0.05, verbose = TRUE) |>
  arrange(source, id)

# tables should align
stopifnot(result_precomputed$id == result_bootstrapped$id)
```

```
# no missing values
stopifnot(is.finite(result_precomputed$pvalue) &
          is.finite(is.finite(result_bootstrapped$pvalue)))

# compare results
plot(result_precomputed$pvalue, result_bootstrapped$pvalue)
abline(0, 1, col=2)

plot(minlog10_fixzero(result_precomputed$pvalue),
     minlog10_fixzero(result_bootstrapped$pvalue))
abline(0, 1, col=2)

summary(minlog10_fixzero(result_precomputed$pvalue) -
        minlog10_fixzero(result_bootstrapped$pvalue))
```

`test_genesets_gsea` *GSEA as implemented in the fgsea R package*

Description

In most cases, it's more convenient to call the more generic `test_genesets` function which also deals with multiple-testing correction (per geneset source)

Usage

```
test_genesets_gsea(
  genesets,
  genelist,
  score_type = NULL,
  parallel_threads = 1L,
  gseaParam = 1,
  nPermSimple = 50000,
  gsea_genelist_col = NULL,
  gsea_scoretype = NULL,
  random_seed = 123
)
```

Arguments

`genesets`	data.frame/tibble with geneset and gene columns
`genelist`	data.frame/tibble with gene and score columns. Should contain columns gene and either pvalue or effectsize, depending on `score_type` parameter
`score_type`	how to compute gene scores? options: "pvalue", "effectsize", "custom". Option "pvalue" uses -log10 transformed values from the pvalue column in `genelist`. Option "effectsize" uses values from the effectsize column in `genelist` as-is. Option "custom" expects 2 additional parameters; `gsea_genelist_col` should be a column name in `genelist` to be used for fGSEA (values used as-is),

gsea_scoretype should be the respective value for the fGSEA scoreType parameter ('pos', 'neg' or 'std')

parallel_threads
: number of threads to use for parallel processing. Set to 0 to automatically select all available processors/cores, set to 1 to disable (default) or to N to use N processes. Note that multiprocessing sometimes breaks on RStudio + Windows, hence this parameter is set to 1 to disable multiprocessing by default for now

gseaParam
: passed to fgsea::fgsea(), from manual: "GSEA parameter value, all gene-level statis are raised to the power of 'gseaParam' before calculation of GSEA enrichment scores.". default = 1. Further comments by fGSEA author at https://github.com/ctlab/fgsea/iss

nPermSimple
: passed to fgsea::fgsea(), from manual: "Number of permutations in the simple fgsea implementation for preliminary estimation of P-values.". default = 50000 in this R package but 1000 by default in fGSEA v1.22.0; we observed much better accuracy in null simulations when increasing this from default 1k to 10k and further minor improvement towards 50k, hence the latter is our default

gsea_genelist_col
: optional, only used for score_type "custom"

gsea_scoretype
: optional, only used for score_type "custom"

random_seed
: the random seed that is passed to set.seed() in order to ensure fgsea results are reproducible. default: 123

Value

input genesets table with results in the "pvalue", "score_type" and "gsea_nes" columns

See Also

test_genesets

---

test_genesets_hypergeometric

Geneset ORA using hypergeometric test

Description

In most cases, it's more convenient to call the more generic test_genesets function which also deals with multiple-testing correction (per geneset source)

It is assumed that the genesets and genelist parameters are in sync, i.e. genesets provided here is the result of the filter_genesets() function (using the same genelist table)

Only genesets with at least 1 significant gene are subjected to statistical testing (e.g. NA is returned for genesets without significant genes)

Usage

```
test_genesets_hypergeometric(genesets, genelist, require_nsignif = 1L)
```

Arguments

genesets	tibble with genesets, must contain columns 'id', 'ngenes' and 'ngenes_signif'
genelist	tibble with genes, must contain column 'signif'. The number of rows in this table (where signif is not NA) is assumed to be the total number of tested genes, the number of rows where signif==TRUE is assumed the total number of significant genes.
require_nsignif	minimum number of 'signif genes' that overlap with a geneset; NA pvalues are returned for genesets with ngenes_signif <= require_nsignif. This function 'prefilters' genesets, so beware that this will influence downstream multiple testing correction. Default is 1

Value

input genesets table with results in the "pvalue" column

See Also

test_genesets

---

treemap_data *Construct tree and treemap data structures from geneset parent/child relations*

Description

refer to the goat::treemap_plot() function for a complete example

Usage

```
treemap_data(
  geneset_ids,
  genesets,
  genesets_test_result,
  simplify = "none",
  toplevel_max_ngenes = NA
)
```

Arguments

geneset_ids	vector of geneset identifiers
genesets	entire geneset table; typically the complete GO database
genesets_test_result	geneset testing results; the output from test_genesets()

simplify — strategy for reducing the genesets returned in the treemap. Options; "leaf_only" (most stringent, returns only leaves in the tree structure) "prune_singletons" (remove parent terms that have exactly 1 child) "pvalue" (remove parent terms where the child term p-value is at least 4 times better) "none" (default; return all significant genesets that are not a "grouping term" in the treemap)

toplevel_max_ngenes — groups in the treemap should not have more than this many genes ('ngenes' in geneset test results). If not set, this defaults to 50% of the total number of unique genes in the geneset test results

Value

data structure needed for `treemap_plot()`

---

treemap_plot *Plot a treemap*

Description

simple wrapper around the treemap R package. To customize this plot, copy/paste its code and tweak parameters as desired

Usage

```
treemap_plot(x, label_group = FALSE)
```

Arguments

x — `treemap_plotdata` data table that was computed by the `treemap_data()` function

label_group — set TRUE to show only group-level labels

Value

a ggplot2 object constructed by `treemap::treemap()`

Examples

```
# note; this example downloads data when first run, and typically takes ~60seconds

# store the downloaded files in the following directory. Here, the temporary file
# directory is used. Alternatively, consider storing this data in a more permanent location.
# e.g. output_dir="~/data/goat" on unix systems or output_dir="C:/data/goat" on Windows
output_dir = tempdir()

## first run the default example from test_genesets() to obtain geneset results
datasets = download_goat_manuscript_data(output_dir)
genelist = datasets$`Wingo 2020:mass-spec:PMID32424284`
```

```
genesets_asis = download_genesets_goatrepo(output_dir)
genesets_filtered = filter_genesets(genesets_asis, genelist)
result = test_genesets(genesets_filtered, genelist, method = "goat",
  score_type = "effectsize", padj_method = "bonferroni", padj_cutoff = 0.05)

# subset GO CC results
x = result |> filter(signif & source == "GO_CC")
tm = treemap_data(
  geneset_ids = x$id,
  genesets = genesets_filtered,
  genesets_test_result = x,
  simplify = "leaf_only" # options: none/leaf_only/prune_singletons/pvalue
)
treemap_plot(tm$treemap_plotdata)
```

Index