Package 'GWENA'

March 30, 2021

Title Pipeline for augmented co-expression analysis

Version 1.0.1

Description The development of high-throughput sequencing led to increased use of co-expression analysis to go beyong single feature (i.e. gene) focus. We propose GWENA (Gene Whole co-Expression Network Analysis) , a tool designed to perform gene co-expression network analysis and explore the results in a single pipeline. It includes functional enrichment of modules of co-expressed genes, phenotypcal association, topological analysis and comparison of networks configuration between conditions.

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 7.1.1

Depends R (>= 4.0.0)

Suggests testthat (>= 2.1.0), knitr (>= 1.25), rmarkdown (>= 1.16), prettydoc (>= 0.3.0), httr (>= 1.4.1), S4Vectors (>= 0.22.1), BiocStyle (>= 2.15.8)

Imports WGCNA (>= 1.67), dplyr (>= 0.8.3), dynamicTreeCut (>= 1.63-1), ggplot2 (>= 3.1.1), gprofiler2 (>= 0.1.6), magrittr (>= 1.5), tibble (>= 2.1.1), tidyr (>= 1.0.0), NetRep (>= 1.2.1), igraph (>= 1.2.4.1), RColorBrewer (>= 1.1-2), purrr (>= 0.3.3), rlist (>= 0.4.6.1), matrixStats (>= 0.55.0), SummarizedExperiment (>= 1.14.1), stringr (>= 1.4.0), methods, graphics, stats, utils

VignetteBuilder knitr

biocViews Software, GeneExpression, Network, Clustering, GraphAndNetwork, GeneSetEnrichment, Pathways, Visualization, RNASeq, Transcriptomics, mRNAMicroarray, Microarray, NetworkEnrichment, Sequencing, GO

BugReports https://github.com/Kumquatum/GWENA/issues

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

git_branch RELEASE_3_12

git_last_commit 59c2afd

git_last_commit_date 2021-02-15

Date/Publication 2021-03-29

Author Gwenaëlle Lemoine [aut, cre] (<https://orcid.org/0000-0003-4747-1937>)

Maintainer Gwenaëlle Lemoine <lemoine.gwenaelle@gmail.com>

R topics documented:

.check_data_expr *Run checks on an object to test if it's a data_expr*

Description

Check an object to be a data.frame or a matrix compatible of genes and samples.

Usage

```
.check_data_expr(data_expr)
```

Arguments

data_expr	matrix or data.frame, expression data with genes as column and samples as row.

Value

Throw an error if doesn't correspond

.check_gost *Run checks on an object to test if it's a gost result*

Description

Take a list that should be a gost result and check if format is good.

Usage

```
.check_gost(gost_result)
```

Arguments

gost_result	list, gprofiler2::gost result

Value

Throw an error if doesn't correspond

.check_module *Run checks on an object to test if it's a module or a list of modules*

Description

Check content of a given object to determine if it's a module or a list of modules, meaning a single vector of characters which are gene names, or a named list of these vectors

Usage

```
.check_module(module, is_list = FALSE)
```

Arguments

module	vector or list, object to test to be a module or list of modules
is_list	boolean, indicate if module must be tested as a single module or a list of modules

Value

Throw an error if doesn't correspond

.check_network *Run checks on an object to test if it's a network*

Description

Check content of a given object to determine if it's a network, meaning a squared matrix of similarity score between genes.

Usage

```
.check_network(network)
```

Arguments

network	matrix or data.frame, object to test to be a network

Value

Throw an error if doesn't correspond

.contingencyTable *Calculate a contigency table of module overlap between datasets*

Description

Calculate a contigency table of module overlap between datasets

Usage

```
.contingencyTable(modAssignments, mods, tiNodelist)
```

Arguments

modAssignments	a list where the first element is the 'moduleAssignments' vector in the discovery dataset, and the second element is the 'moduleAssignments vector in the test dataset.
mods	the 'modules' vector for the discovery dataset.
tiNodelist	a vector of node IDs in the test dataset.

Value

A list containing a contigency table, a vector of the proportion of nodes present in the test dataset for each module, a vector containing the number of nodes present in the test dataset for each module, a vector of the node names present in both the discovery and test datasets, a vector of modules that are both requested and have nodes present in the test dataset, and the modAssignments vector containing only nodes present in the test dataset.

.cor_func_match *Match a correlation function based on a name*

Description

Translate a function name into an R function.

Usage

```
.cor_func_match(cor_func = c("pearson", "spearman", "bicor"))
```

Arguments

cor_func	string of the name of the correlation to be use

Value

A function corresponding to the correlation required

associate_phenotype *Modules phenotpic association*

Description

Compute the correlation between all modules and the phenotypic variables

Usage

```
associate_phenotype(eigengenes, phenotypes, id_col = NULL)
```

Arguments

eigengenes	matrix or data.frame, eigengenes of the modules. Provided by the output of modules_detection.
phenotypes	matrix or data.frame, phenotypes for each sample to associate.
id_col	string or vector of string, optional name of the columns containing the common id between eigengenes and phenotypes.

Value

A list of two data.frames : associations modules/phenotype and p.values associated to this associations

Examples

```
eigengene_mat <- data.frame(mod1 = rnorm(20, 0.1, 0.2),
mod2 = rnorm(20, 0.2, 0.2))
phenotype_mat <- data.frame(phenA = sample(c("X", "Y", "Z"), 20,
                            replace = TRUE),
                            phenB = sample(c("U", "V"), 20, replace = TRUE),
                            stringsAsFactors = FALSE)
association <- associate_phenotype(eigengene_mat, phenotype_mat)
```

bio_enrich *Modules enrichment*

Description

Enrich genes list from modules.

Usage

```
bio_enrich(module, custom_gmt = NULL, ...)
```

Arguments

module	vector or list, vector of gene names representing a module or a named list of this modules.
custom_gmt	string or list, path to a gmt file or a list of these path.
...	any other parameter you can provide to gprofiler2::gost function.

Value

A gprofiler2::gost output, meaning a named list containing a 'result' data.frame with enrichement information on the differents databases and custom gmt files, and a 'meta' list containing informations on the input args, the version of gost, timestamp, etc. For more detail, see ?gprofiler2::gost.

Examples

```
custom_path <- system.file("extdata", "h.all.v6.2.symbols.gmt",
                           package = "GWENA", mustWork = TRUE)

single_module <- c("BIRC3", "PMAIP1", "CASP8", "JUN", "BCL2L11", "MCL1",
                   "IL1B", "SPTAN1", "DIABLO", "BAX", "BIK", "IL1A", "BID",
                   "CDKN1A", "GADD45A")
single_module_enriched <- bio_enrich(single_module, custom_path)

multi_module <- list(mod1 = single_module,
                     mod2 = c("TAF1C", "TARBP2", "POLH", "CETN2", "POLD1",
                              "CANT1", "PDE4B", "DGCR8", "RAD51", "SURF1",
                              "PNP", "ADA", "NME3", "GTF3C5", "NT5C"))
multi_module_enriched <- bio_enrich(multi_module, custom_path)
```

---

build_graph_from_sq_mat

Return graph from squared matrix network

Description

Takes a squared matrix containing the pairwise similarity scores for each gene and return a igraph object.

Usage

```
build_graph_from_sq_mat(sq_mat)
```

Arguments

sq_mat	matrix or data.frame, squared matrix representing

Value

An igraph object

Examples

```
mat <- matrix(runif(40*40), 40)
build_graph_from_sq_mat(mat)
```

`build_net` *Network building by co-expression score computation*

Description

Compute the adjacency matrix, then the TOM to build the network. Than detect the modules by hierarchical clustering and thresholding

Usage

```
build_net(
  data_expr,
  fit_cut_off = 0.9,
  cor_func = c("pearson", "spearman", "bicor", "other"),
  your_func = NULL,
  power_value = NULL,
  block_size = NULL,
  stop_if_no_fit = FALSE,
  network_type = c("unsigned", "signed", "signed hybrid"),
 tom_type = c("unsigned", "signed", "signed Nowick", "unsigned 2", "signed 2", "none"),
  keep_matrices = c("none", "cor", "adja", "both"),
  n_threads = NULL,
  ...
)
```

Arguments

`data_expr`	matrix or data.frame or SummarizedExperiment, expression data with genes as column and samples as row.
`fit_cut_off`	float, cut off by which R^2 (coefficient of determination) will be thresholded. Must be in]0;1[.
`cor_func`	string, name of the correlation function to be used. Must be one of "pearson", "spearman", "bicor", "other". If "other", your_func must be provided
`your_func`	function returning correlation values. Final values must be in [-1;1]
`power_value`	integer, power to be applied to the adjacency matrix. If NULL, will be estimated by trying different power law fitting.
`block_size`	integer, size of blocks by which operations can be proceed. Helping if working with low capacity computers. If null, will be estimated.
`stop_if_no_fit`	boolean, does not finding a fit above fit_cut_off should stop process, or just print a warning and return the highest fitting power.
`network_type`	string, type of network to be used. Either "unsigned", "signed", "signed hybrid". See details.

tom_type	string, type of the topological overlap matrix to be computed. Either "none", "unsigned", "signed", "signed Nowick", "unsigned 2", "signed 2" and "signed Nowick 2". See detail at TOMsimilarityFromExpr.
keep_matrices	string, matrices to keep in final object. Can be one of "none", "cor", "adja", "both". It is usefull to keep both if you plant to use compare_conditions.
n_threads	integer, number of threads that can be used to paralellise the computing
...	any other parameter compatible with adjacency.fromSimilarity

Value

list containing network matrix, metadata of input parameters and power fitting information.

Examples

```
net <- build_net(kuehne_expr[, seq_len(350)], n_threads = 1)
```

---

compare_conditions *Compare modules topology between conditions*

Description

Take modules built from multiples conditions and search for preservation, non-preservation or one of them, against one or mutliple conditions of reference. Use 7 topological features to perform the differents test, and use permutation to validate results.

Usage

```
compare_conditions(
  data_expr_list,
  adja_list,
  cor_list = NULL,
  modules_list,
  ref = names(data_expr_list)[1],
  test = NULL,
  cor_func = c("pearson", "spearman", "bicor", "other"),
  your_func = NULL,
  n_perm = 10000,
  test_alternative_hyp = c("greater", "less", "two.sided"),
  pvalue_th = 0.01,
  n_threads = NULL,
  ...
)
```

Arguments

data_expr_list	list of matrix or data.frame or SummarizedExperiment, list of expression data by condition, with genes as column and samples as row.
adja_list	list of adjacency matrices, list of square tables by condition, representing connectivity between each genes as returned by build_net.

`cor_list` list of matrices and/or data.frames, list of square tables by condition, representing correlation between each gene. Must be the same used to create networks in `build_net`. If NULL, will be re-calculated according to `cor_func`.

`modules_list` list of modules or nested list of modules, list of modules in one condition (will be considered as the one from reference) or a condition named list with list of modules built in each one.

`ref` string or vector of strings, condition(s) name to be used as reference for permutation tests, or "cross comparison" if you want to compare each condition with the other as reference. Default will be the name of the first element in data_expr_list.

`test` string or vector of strings, condition(s) name to be tested for permutation tests. If NULL, all conditions except these in ref will be taken. If ref is set to "cross comparison", any test specified will be ignored.

`cor_func` string, name of the correlation function to be used. Must be one of "pearson", "spearman", "bicor", "other". If "other", your_func must be provided

`your_func` function returning correlation values. Final values must be in [-1;1]

`n_perm` integer, number of permutation, meaning number of random gene name reassignment inside network to compute all tests and statistics for module comparison between condition.

`test_alternative_hyp` string, either "greater", "less" or "two.sided". Alternative hypothesis (H1) used for the permutation test. Determine if the metrics computed on permuted values are expected to be greater, less or both than the observed ones. More details:`modulePreservation`

`pvalue_th` decimal, threshold of pvalue below which test_alternative_hyp is considered significant. If "two.sided", then pvalue_th is splitted in two for each side (preserved/not preserved).

`n_threads` integer, number of threads that can be used to paralellise the computing

`...` any other parameter compatible with `modulePreservation`

Details

Conditions will be based on names of data_expr_list. Please do not use numbers for conditions names as modules are often named this way

The final comparison output is a combination of the permutation test and the Z summary statistic. Comparison value is set to "preserved" when both return "preserved", "moderately preserved" when Z summary return it, "unpreserved" when permutation test is not significant and Z summary return "unpreserved", and "inconclusive" when the two values are opposite

To avoid recalculation, correlations matrices can be obtain by setting `keep_cor_mat` in `build_net` to TRUE.

Description of the 7 topological features used for preservation testing is available in `modulePreservation`.

Value

A nested list where first element is each ref provided, second level each condition to test, and then elements containing information on the comparison. See NetRep::modulePreservation() for more detail.

Examples

```
expr_by_cond <- list(cond1 = kuehne_expr[1:24, 1:350],
                     cond2 = kuehne_expr[25:48, 1:350])
net_by_cond <- lapply(expr_by_cond, build_net, cor_func = "spearman",
                      n_threads = 1, keep_matrices = "both")
mod_by_cond <- mapply(detect_modules, expr_by_cond,
                      lapply(net_by_cond, `[[`, "network"),
                      MoreArgs = list(detailled_result = TRUE),
                      SIMPLIFY = FALSE)
comparison <- compare_conditions(expr_by_cond,
                                 lapply(net_by_cond, `[[`, "adja_mat"),
                                 lapply(net_by_cond, `[[`, "cor_mat"),
                                 lapply(mod_by_cond, `[[`, "modules"),
                                 n_perm = 100)
```

`detect_modules` *Modules detection in a network*

Description

Detect the modules by hierarchical clustering .

Usage

```
detect_modules(
  data_expr,
  network,
  min_module_size = min(20, ncol(data_expr)/2),
  clustering_th = NULL,
  merge_close_modules = TRUE,
  merge_threshold = 0.75,
  detailled_result = TRUE,
  pam_respects_dendro = FALSE,
  ...
)
```

Arguments

`data_expr`	matrix or data.frame or SummarizedExperiment, expression data with genes as column and samples as row.
`network`	matrix or data.frame, strengh of gene co-expression (edge values).
`min_module_size`	integer, lowest number of gene allowed in a module. If none provided, estimated.
`clustering_th`	float, threshold to be used by the clustering method. For now `cutreeDynamic`.
`merge_close_modules`	boolean, does closest modules (based on eigengene) should be merged together.

`merge_threshold`
: float, eigengenes correlation value over which close modules will be merged. Must be in]0;1[. See `mergeCloseModules`

`detailled_result`
: boolean, does pre-merge modules (if applicable) and dendrogram included in output.

`pam_respects_dendro`
: boolean, If TRUE, the Partitioning Around Medoids (PAM) stage will respect the dendrogram in the sense that objects and small clusters will only be assigned to clusters that belong to the same branch that the objects or small clusters being assigned belong to.

`...`
: any other parameter compatible with `mergeCloseModules`

Value

list containing modules detected, modules_eigengenes, and if asked for, modules pre-merge and dendrograms of genes and merged modules

Examples

```
df <- kuehne_expr[1:24, 1:350]
net <- build_net(df, n_threads = 1)
detect_modules(df, net$network)
```

`filter_low_var` *Filtering genes with low variability*

Description

Remove low variating genes based on the percentage given and the type of variation specified.

Usage

```
filter_low_var(data_expr, pct = 0.8, type = c("mean", "median", "mad"))
```

Arguments

`data_expr`
: matrix or data.frame or SummarizedExperiment, table of expression values (either microarray or RNA-seq), with genes as column and samples as row

`pct`
: float, percentage of gene to keep, value must be in]0;1[

`type`
: string, function name used for filtration. Should be either "mean", "median", or "mad"

Value

A data.frame of filtered genes

Examples

```
df <- matrix(abs(rnorm(15*45)), 15)
colnames(df) <- paste0("gene_", seq_len(ncol(df)))
rownames(df) <- paste0("sample_", seq_len(nrow(df)))
df_filtered <- filter_low_var(df)
```

`filter_RNA_seq` *Filtering of low counts*

Description

Keeping genes with at least one sample with count above min_count in RNA-seq data.

Usage

```
filter_RNA_seq(
  data_expr,
  min_count = 5,
  method = c("at least one", "mean", "all")
)
```

Arguments

`data_expr`	matrix or data.frame or SummarizedExperiment, table of expression values (either microarray or RNA-seq), with genes as column and samples as row.
`min_count`	integer, minimal number of count to be considered in method.
`method`	string, name of the method for filtering. Must be one of "at least one", "mean", or " all"

Details

Low counts in RNA-seq can bring noise to gene co-expression module building, so filtering them help to improve quality.

Value

A data.frame of filtered genes

Examples

```
df <- matrix(abs(rnorm(15*45)), 15) * 3
colnames(df) <- paste0("gene_", seq_len(ncol(df)))
rownames(df) <- paste0("sample_", seq_len(nrow(df)))
df_filtered <- filter_RNA_seq(df)
```

`get_fit.cor` *Calculating best fit of a power low on correlation matrix computed on expression data*

Description

Adjust a correlation matrix depending of the type of network, then try to parameter a power law for best fit

Usage

```
get_fit.cor(
  cor_mat,
  fit_cut_off = 0.9,
  network_type = c("unsigned", "signed", "signed hybrid"),
  block_size = NULL,
  ...
)
```

Arguments

`cor_mat`	matrix or data.frame of genes correlation.
`fit_cut_off`	float, cut off by which R^2 (coefficient of determination) will be thresholded. Must be in]0;1[.
`network_type`	string giving type of network to be used. Either "unsigned", "signed", "signed hybrid". See details.
`block_size`	integer giving size of blocks by which operations can be proceed. Helping if working with low capacity computers. If null, will be estimated.
`...`	any other parameter compatible with `pickSoftThreshold.fromSimilarity`

Details

network_type indicate which transformation will be applied on the correlation matrix to return the similarity score.

signed will modify the range [-1;1] to [0.5;1.5] (because of log10 beeing used for scale free index computation)

unsigned will return absolute value (moving from [-1;1] to [0;1])

signed hybrid will replace all negative values by 0 (moving from [-1;1] to [0;1])

Value

A list containing power of the law for best fit, fit table, and metadata about the arguments used.

Examples

```
get_fit.cor(cor_mat = cor(kuehne_expr[, seq_len(100)]))
```

get_fit.expr *Calculating best fit of a power low on expression data*

Description

Computes correlation matrix of the gene expression data, adjust it depending of the type of network, then try to parameter a power law for best fit

Usage

```
get_fit.expr(
  data_expr,
  fit_cut_off = 0.9,
  cor_func = c("pearson", "spearman", "bicor", "other"),
  your_func = NULL,
  network_type = c("unsigned", "signed", "signed hybrid"),
  block_size = NULL,
  ...
)
```

Arguments

data_expr	matrix or data.frame or SummarizedExperiment, expression data with genes as column and samples as row.
fit_cut_off	float, cut off by which R^2 (coefficient of determination) will be thresholded. Must be in]0;1[.
cor_func	string specifying correlation function to be used. Must be one of "pearson", "spearman", "bicor", "other". If "other", your_func must be provided
your_func	function returning correlation values. Final values must be in [-1;1]
network_type	string giving type of network to be used. Either "unsigned", "signed", "signed hybrid". See details.
block_size	integer giving size of blocks by which operations can be proceed. Helping if working with low capacity computers. If null, will be estimated.
...	any other parameter compatible with pickSoftThreshold.fromSimilarity

Details

network_type indicate which transformation will be applied on the correlation matrix to return the similarity score.

signed will modify the range [-1;1] to [0.5;1.5] (because of log10 beeing used for scale free index computation)

unsigned will return absolute value (moving from [-1;1] to [0;1])

signed hybrid will replace all negative values by 0 (moving from [-1;1] to [0;1])

Value

A list containing power of the law for best fit, fit table, and metadata about the arguments used.

Examples

```
get_fit.expr(kuehne_expr[, seq_len(100)])
```

`get_hub_degree` *Determine hub genes based on degree*

Description

Remove edges from the graph which value is under weight_th then compute degree of each node (gene). Hub gene are genes whose degree value is above average degree value of the thresholded network.

Usage

```
get_hub_degree(network, modules = NULL, weight_th = 0.2)
```

Arguments

`network`	matrix or data.frame, square table representing connectivity between each genes as returned by build_net. Can be whole network or a single module.
`modules`	list, modules defined as list of gene vectors. If null, network is supposed to be the whole network or an already split module
`weight_th`	decimal, weight threshold under or equal to which edges will be removed

Details

GWENA natively build networks using WGCNA. These networks are complete in a graph theory sens, meaning all nodes are connected to each other. Therefore a threshold need to be applied so degree of all nodes isn't the same.

Value

A list of vectors, or single vector of gene names

Examples

```
mat <- matrix(runif(40*40), 40)
colnames(mat) <- paste0("gene_", seq_len(ncol(mat)))
rownames(mat) <- paste0("gene_", seq_len(nrow(mat)))
get_hub_degree(mat)
```

get_hub_genes *Determine hub genes inside each module*

Description

Return genes considered as hub genes inside each module of a network following the selected method. Method will be lauched with default parameters. If specific parameters desired, please use directly the function `get_hub_...` itself.

Usage

```
get_hub_genes(
  network,
  modules = NULL,
  method = c("highest connectivity", "superior degree", "Kleinberg's score")
)
```

Arguments

`network`	matrix or data.frame, square table representing connectivity between each genes as returned by build_net. Can be whole network or a single module.
`modules`	list, modules defined as list of gene vectors. If null, network is supposed to be the whole network or an already split module
`method`	string, name of the method to be used for hub gene detection. See details.

Details

highest connectivity Select the top n (n depending on parameter given) highest connected genes. Similar to WGCNA::chooseTopHubInEachModule.

superior degree Select genes which degree is greater than average connection degree of the network. Definition from network theory.

Kleinberg's score Select genes which Kleinberg's score superior to provided threshold.

Value

A list of vectors representing hub genes, by module

Examples

```
mat <- matrix(runif(40*40), 40)
colnames(mat) <- paste0("gene_", seq_len(ncol(mat)))
rownames(mat) <- paste0("gene_", seq_len(nrow(mat)))
get_hub_genes(mat)
```

get_hub_high_co *Determine hub genes based on connectivity*

Description

Compute connectivity of each gene by module if provided or for whole network if not, and return the top_n highest connected ones.

Usage

```
get_hub_high_co(network, modules = NULL, top_n = 5)
```

Arguments

network	matrix or data.frame, square table representing connectivity between each genes as returned by build_net. Can be whole network or a single module.
modules	list, modules defined as list of gene vectors. If null, network is supposed to be the whole network or an already split module
top_n	integer, number of genes to be considered as hub genes

Value

A list of vectors, or single vector of gene names

Examples

```
mat <- matrix(runif(40*40), 40)
colnames(mat) <- paste0("gene_", seq_len(ncol(mat)))
rownames(mat) <- paste0("gene_", seq_len(nrow(mat)))
get_hub_high_co(mat)
```

get_hub_kleinberg *Determine hub genes based on Kleinberg's score*

Description

Compute Kleinberg's score (defined as the principal eigenvector of A*t(A), where A is the similarity matrix of the graph) of each gene by module if provided or for whole network if not, and return the top_n highest ones.

Usage

```
get_hub_kleinberg(network, modules = NULL, top_n = 5, k_th = NULL)
```

Arguments

network	matrix or data.frame, square table representing connectivity between each genes as returned by build_net. Can be whole network or a single module.
modules	list, modules defined as list of gene vectors. If null, network is supposed to be the whole network or an already split module
top_n	integer, number genes to be considered as hub genes
k_th	decimal, Kleinberg's score threshold above or equal to which genes are considered as hubs

Details

If you provide a top_n value, you can't provide a k_th value and vice versa. If none of them is provided, top_n = 5. For more information on Kleinberg's score, look at hub_score from igraph.

Value

A list of vectors, or single vector of gene names

Examples

```
mat <- matrix(runif(40*40), 40)
colnames(mat) <- paste0("gene_", seq_len(ncol(mat)))
rownames(mat) <- paste0("gene_", seq_len(nrow(mat)))
get_hub_degree(mat)
get_hub_kleinberg(mat, top_n = NULL, k_th = 0.9)
```

---

gtex_expr *Transcriptomic muscle data from GTEx consorsium RNA-seq data*

Description

A subset of GTEx RNA-seq dataset containing read counts collapsed to gene level.

Usage

```
gtex_expr
```

Format

A data frame with 50 rows (samples) and 15000 columns (genes)

Source

https://gtexportal.org/home/datasets

gtex_traits *Traits data linked to samples in transcriptomic data from GTEx*

Description

A dataset containing phenotypes of donors. From public data. Note: protected data contain more information but require dbGap accessh (see https://gtexportal.org/home/protectedDataAccess).

Usage

```
gtex_traits
```

Format

A data frame with 50 rows (samples) and 4 columns :

SUBJID Subject ID, GTEx Public Donor ID

SEX Sex, donor's Identification of sex based upon self-report : 1=Male, 2=Female

AGE Age range, elapsed time since birth in years

DTHHRDY Hardy Scale : 0=Ventilator Case, 1=Violent and fast death, 2=Fast death of natural causes, 3=Intermediate death, 4=Slow death)

Source

https://gtexportal.org/home/datasets

is_data_expr *Determine if an object is a data_expr in sens of GWENA*

Description

Check an object to be a data.frame or a matrix compatible of genes and samples.

Usage

```
is_data_expr(data_expr)
```

Arguments

data_expr	matrix or data.frame, expression data with genes as column and samples as row.

Value

list, a boolean as first element and in second element NULL or the reason why boolean is set to FALSE

Examples

```
expr <- matrix(runif(15*40), 15)
colnames(expr) <- paste0("gene_", seq_len(ncol(expr)))
rownames(expr) <- paste0("gene_", seq_len(nrow(expr)))
is_data_expr(expr)
```

is_gost *Determine if an object is a gost object*

Description

Check content of a given object to determine if it's a gost object

Usage

```
is_gost(gost_result)
```

Arguments

gost_result	list, gprofiler2::gost result

Value

list, a boolean as first element and in second element NULL or the reason why boolean is set to FALSE

Examples

```
single_module <- c("BIRC3", "PMAIP1", "CASP8", "JUN", "BCL2L11", "MCL1",
                   "IL1B", "SPTAN1", "DIABLO", "BAX", "BIK", "IL1A", "BID",
                   "CDKN1A", "GADD45A")
single_module_enriched <- bio_enrich(single_module)
is_gost(single_module_enriched)
```

is_module *Determine if an object is a module or a list of modules*

Description

Check content of a given object to determine if it's a module or a list of modules, meaning a single vector of characters which are gene names, or a named list of these vectors

Usage

```
is_module(module, is_list = FALSE)
```

Arguments

module	vector or list, object to test to be a module or list of modules
is_list	boolean, indicate if module must be tested as a single module or a list of modules

Value

list, a boolean as first element and in second element NULL or the reason why boolean is set to FALSE

Examples

```
single_module <- c("BIRC3", "PMAIP1", "CASP8", "JUN", "BCL2L11", "MCL1",
                   "IL1B", "SPTAN1", "DIABLO", "BAX", "BIK", "IL1A", "BID",
                   "CDKN1A", "GADD45A")
is_module(single_module)

multi_module <- list(mod1 = single_module,
                     mod2 = c("TAF1C", "TARBP2", "POLH", "CETN2", "POLD1",
                     "CANT1", "PDE4B", "DGCR8", "RAD51", "SURF1", "PNP",
                     "ADA", "NME3", "GTF3C5", "NT5C"))
is_module(multi_module$modules, is_list = TRUE)
```

`is_network` *Determine if an object is a network*

Description

Check content of a given object to determine if it's a network, meaning a squared matrix of similarity score between genes.

Usage

```
is_network(network)
```

Arguments

`network`	matrix or data.frame, object to test to be a network

Value

list, a boolean as first element and in second element NULL or the reason why boolean is set to FALSE

Examples

```
net <- matrix(runif(40*40), 40)
colnames(net) <- paste0("gene_", seq_len(ncol(net)))
rownames(net) <- paste0("gene_", seq_len(nrow(net)))
is_network(net)
```

join_gost *Join gprofiler2::gost results*

Description

Takes list of gprofiler2::gost results and join them. Usefull to join results of gprofiler2::gost with custom gmt to other gprofiler2::gost results.

Usage

```
join_gost(gost_result)
```

Arguments

gost_result	list of gprofiler2::gost result

Details

First element of the list is taken as reference for checks on gost_result elements compatibility. If warnings returned, value from reference will be used. Also, timestamp is set to timestamp of the join

Value

A gprofiler2::gost result

Examples

```
query <- c("ENSG00000184349", "ENSG00000158955", "ENSG00000091140",
           "ENSG00000163114", "ENSG00000163132", "ENSG00000019186")
g1 <- gprofiler2::gost(query, sources = "GO")
g2 <- gprofiler2::gost(query, sources = "REAC")
gj <- join_gost(list(g1,g2))
```

kuehne_expr *Transcriptomic data from the Kuehne et al. publication*

Description

A dataset containing the expression levels collapsed to the gene level. Obtained from script provided in additional data n°10 runned on GSE85358 and reduced from probe to gene by WGCNA::collapseRows with median as fucntion.

Usage

```
kuehne_expr
```

Format

A data frame with 48 rows (samples) and 15801 columns (genes).

Source

https://bmcgenomics.biomedcentral.com/articles/10.1186/s12864-017-3547-3

`kuehne_traits` *Traits data linked to samples in transcriptomic data from the Kuehne et al. publication*

Description

A dataset containing the phenotype of the donors and technical information about the experiment

Usage

```
kuehne_traits
```

Format

A data frame with 48 rows (samples) and 5 columns :

Slide Reference number of the microarray's slide.

Array Array number, 8 by slide usually

Exp Experiment number

Condition Either old (between 55 and 66 years old) or young (between 20 to 25 years old)

Age Real age of the donor

Source

https://bmcgenomics.biomedcentral.com/articles/10.1186/s12864-017-3547-3

`plot_comparison_stats` *Heatmap of comparison statistics*

Description

Plot heatmap of p values for the module comparison statistics evaluated through the permutation test.

Usage

```
plot_comparison_stats(
  comparison_pvalues,
  pvalue_th = 0.05,
  low_color = "#031643",
  pvalue_th_color = "#A0A3D3",
  unsignificant_color = "#FFFFFF"
)
```

Arguments

comparison_pvalues	matrix or data.frame, table containing the p values for the statistics on each module
pvalue_th	decimal, threshold of pvalue below which statistics are considered as significant
low_color, pvalue_th_color, unsignificant_color	string, color to use as lower, middle, and higher end of the legend. Can either be the color name or hexadecimal code (e.g.: "red" or "#FF1234")

Value

A ggplot object representing a heatmap of the comparison statistics for each module

Examples

```
df <- data.frame(avg.weight = abs(rnorm(4, 0.1, 0.1)),
                 coherence = abs(rnorm(4, 0.1, 0.1)),
                 cor.cor = abs(rnorm(4, 0.1, 0.1)),
                 cor.degree = abs(rnorm(4, 0.1, 0.1)),
                 cor.contrib = abs(rnorm(4, 0.1, 0.1)),
                 avg.cor = abs(rnorm(4, 0.1, 0.1)),
                 avg.contrib = abs(rnorm(4, 0.1, 0.1)))
plot_comparison_stats(df)
```

---

plot_enrichment *Plot module from bio_enrich*

Description

Wrapper of the gprofiler2::gostplot function. Adding support of colorblind palet and selection of subsets if initial multiple query, and/or sources to plot.

Usage

```
plot_enrichment(
  enrich_output,
  modules = "all",
  sources = "all",
  colorblind = TRUE,
  custom_palette = NULL,
  ...
)
```

Arguments

enrich_output	list, bio_enrich result which are in fact gprofiler2::gost output.
modules	string or vector of characters designing the modules to plot. "all" by default to plot every module.
sources	string or vector of characters designing the sources to plot. "all" by default to plot every source.

colorblind	boolean, indicates if a colorblind friendly palette should be used.
custom_palette	vector of character, colors to be used for plotting.
...	any other parameter you can provide to gprofiler2::gostplot.

Details

Note: The colorblind friendly palette is limited to maximum 8 colors, therefore 8 sources of enrichment.

Value

A plotly object representing enrichment for specified modules

Examples

```
custom_path <- system.file("extdata", "h.all.v6.2.symbols.gmt",
                           package = "GWENA", mustWork = TRUE)
multi_module <- list(mod1 = c("BIRC3", "PMAIP1", "CASP8", "JUN", "BCL2L11",
                              "MCL1", "IL1B", "SPTAN1", "DIABLO", "BAX",
                              "BIK", "IL1A", "BID", "CDKN1A", "GADD45A"),
                     mod2 = c("TAF1C", "TARBP2", "POLH", "CETN2", "POLD1",
                              "CANT1", "PDE4B", "DGCR8", "RAD51", "SURF1",
                              "PNP", "ADA", "NME3", "GTF3C5", "NT5C"))
multi_module_enriched <- bio_enrich(multi_module, custom_path)
plot_enrichment(multi_module_enriched)
```

---

plot_expression_profiles

Modules expression profiles

Description

Plot expression profiles for all modules with eigengene highlighted

Usage

```
plot_expression_profiles(data_expr, modules, ...)
```

Arguments

data_expr	matrix or data.frame or SummarizedExperiment, expression data with genes as column and samples as row.
modules	vector, id (whole number or string) of modules associated to each gene.
...	additional parameters to pass to ggplot2::theme

Value

A ggplot representing expression profile and eigengene by module

Examples

```
df <- kuehne_expr[1:24, 1:350]
net <- build_net(df, n_threads = 1)
detection <- detect_modules(df, net$network, detailled_result = TRUE)
plot_expression_profiles(df, detection$modules)
```

`plot_module` *Plot co-expression network*

Description

Display a graph representing the co-expression network and different informations like hubs, enrichments

Usage

```
plot_module(
  graph_module,
  hubs = NULL,
  lower_weight_th = NULL,
  upper_weight_th = NULL,
  enrichment = NULL,
  title = "Module",
  degree_node_scaling = TRUE,
  node_scaling_min = 1,
  edge_scaling_min = 0.2,
  node_scaling_max = 6,
  edge_scaling_max = 1,
  nb_row_legend = 6,
  layout = "auto",
  zoom = 1,
  vertex.label.cex = 0.7,
  vertex.label.color = "gray20",
  vertex.label.family = "Helvetica",
  edge.color = "gray70",
  vertex.frame.color = "white",
  vertex.color = "gray60",
  vertex.label.dist = 1,
  legend_cex = 0.8,
  window_x_min = -1,
  window_x_max = 1,
  window_y_min = -1,
  window_y_max = 1,
  legend = TRUE,
  ...
)
```

Arguments

graph_module	igraph object, module to plot.
hubs	character vector or numeric vector with names, optionnal, vector of gene names or vector of numeric values named with gene names.
lower_weight_th, upper_weight_th	decimal, weight threshold below or above which edges will be removed.
enrichment	list, representing a gost object.
title	string, main title that will be displayed on the plot.
degree_node_scaling	boolean, indicates if node size should represent the degree of this node.
node_scaling_min, node_scaling_max	integer, if degree_node_scaling is TRUE, it is the min/max size of the node, else it is the exact size of all node.
edge_scaling_min, edge_scaling_max	integer, min/max width of the edge
nb_row_legend	integer, number of levels in the legend.
layout	numeric matrix or function or string, numeric matrix for nodes coordinates, or function for layout, or name of a layout function available in igraph. Default "auto" will choose the best layout depending on the graph. For more information, see igraph.plotting.
zoom	integer, scaling factor by which it's possible to have compact graph (< 1) or larger graph (> 1) display.
vertex.label.cex, legend_cex	float, font size for vertex labels. It is interpreted as a multiplication factor of some device-dependent base font size. If 0, no labels displayed.
vertex.label.color, edge.color, vertex.frame.color, vertex.color	character and/or integer vector , color of the labels. It may either contain integer values, named colors or RGB specified colors with three or four bytes. All strings starting with '#' are assumed to be RGB color specifications. It is possible to mix named color and RGB colors.
vertex.label.family	character, font family to be used for vertex labels.
vertex.label.dist	integer, distance of the label from the center of the vertex. If it is 0 then the label is centered on the vertex. If it is 1 then the label is displayed beside the vertex.
window_x_min	decimal, value for the bottom limit of the window.
window_x_max	decimal, value for the top limit of the window.
window_y_min	decimal, value for the left limit of the window.
window_y_max	decimal, value for the right limit of the window.
legend	boolean, indicates if the legend should be plotted.
...	any other parameter compatible with the plot.igraph function.

Details

Take care of you intend to compare modules' graphs, the same size of node will not correspond to the same values because of the scaling.

Value

matrix, layout of the graph as a two column matrix (x, y)

Examples

```
mat <- matrix(runif(40*40), 40)
g <- build_graph_from_sq_mat(mat)
plot_module(g)
```

`plot_modules_merge` *Modules merge plot*

Description

Plot a bipartite graph to see in which modules all modules have been merged

Usage

```
plot_modules_merge(
  modules_premerge,
  modules_merged,
  zoom = 1,
  vertex_size = 6,
  vertex_label_color = "gray20",
  vertex_label_family = "Helvetica",
  vertex_label_cex = 0.8,
  vertex_color = "lightskyblue",
  vertex_frame_color = "white",
  window_x_min = -1,
  window_x_max = 1,
  window_y_min = -1,
  window_y_max = 1,
  ...
)
```

Arguments

`modules_premerge`
: vector, id (whole number or string) of module before merge associated to each gene.

`modules_merged`
: vector, id (whole number or string) of module after merge associated to each gene.

`zoom`
: decimal, value to which the display will be increased/decreased.

`vertex_size`
: integer, size of the vertices.

`vertex_label_color`, `vertex_color`, `vertex_frame_color`
: string, name of the color or hexadecimal code.

`vertex_label_family`
: string, font family name.

vertex_label_cex
: decimal, value for font size.

window_x_min
: decimal, value for the bottom limit of the window.

window_x_max
: decimal, value for the top limit of the window.

window_y_min
: decimal, value for the left limit of the window.

window_y_max
: decimal, value for the right limit of the window.

...
: additional arguments to be passed to igraph::plot.igraph().

Details

Both vectors must be in the same gene order before passing them to the function. No check is applied on this.

Value

The layout of the plot

Examples

```
df <- kuehne_expr[1:24, 1:350]
net <- build_net(df, n_threads = 1)
detection <- detect_modules(df, net$network, detailled_result = TRUE)
plot_modules_merge(modules_premerge = detection$modules_premerge,
                   modules_merged = detection$modules)
```

---

plot_modules_phenotype

Heatmap of modules phenotpic association

Description

Plot a heatmap of the correlation between all modules and the phenotypic variables and the p value associated

Usage

```
plot_modules_phenotype(modules_phenotype, pvalue_th = 0.05, ...)
```

Arguments

modules_phenotype
: list, data.frames of correlation and pvalue associated

pvalue_th
: float, threshold in]0;1[under which module will be considered as significantly associated

...
: any other parameter you can provide to ggplot2::theme

Value

A ggplot object representing a heatmap with phenotype association and related pvalues

Examples

```
eigengene_mat <- data.frame(mod1 = rnorm(20, 0.1, 0.2),
mod2 = rnorm(20, 0.2, 0.2))
phenotype_mat <- data.frame(phenA = sample(c("X", "Y", "Z"), 20,
                            replace = TRUE),
                            phenB = sample(c("U", "V"), 20,
                            replace = TRUE),
                            stringsAsFactors = FALSE)
association <- associate_phenotype(eigengene_mat, phenotype_mat)
plot_modules_phenotype(association)
```

`quiet` *Muting a function*

Description

Prevent a function to output multiple message. Source: https://r.789695.n4.nabble.com/Suppressing-output-e-g-from-cat-td859876.html

Usage

```
quiet(func)
```

Arguments

`func`	Function who need to be muted.

Value

Nothing, just mute the called function

`z_summary` *Calculating Z summary*

Description

Use the topological metrics and permutations from output of `modulePreservation` to compute a Z summary (a composite preservation statistic) as defined by https://doi.org/10.1371/journal.pcbi.1001057

Usage

```
z_summary(observed_stat, permutations_array)
```

Arguments

`observed_stat`	matrix, bidimensional matrix containing the topological matrix computed for each module by `modulePreservation` (the element `observed`). Modules are in row, metrics are in column.
`permutations_array`	matrix, tridimensional matrix containing the topological matrix computed for each module by `modulePreservation` (the element `observed`). Modules are in dim 1, metrics are in dim 2, permutations are in dim 3.

Details

The original Zsummary composite preservation statistic was defined by Langfelder et al. (2011). However this method use the metric from modulePreservation since they it handle better large and multiple testing correction.

Value

A named vector of the z summary statistic with the module id as name.

Examples

```
expr_by_cond <- list(cond1 = kuehne_expr[1:24, 1:350],
                     cond2 = kuehne_expr[25:48, 1:350])
net_by_cond <- lapply(expr_by_cond, build_net, cor_func = "spearman",
                      n_threads = 1, keep_matrices = "both")

mods_labels <- setNames(
  sample(1:6, 350, replace = TRUE,
         prob = c(0.05, 0.4, 0.25, 0.15, 0.1, 0.05)),
  colnames(expr_by_cond$cond1))

netrep_res <- NetRep::modulePreservation(
  network = lapply(net_by_cond, `[[`, "adja_mat"),
  data = lapply(expr_by_cond, as.matrix),
  correlation = lapply(net_by_cond, `[[`, "cor_mat"),
  moduleAssignments = mods_labels,  nPerm = 100)

z_summary(netrep_res$observed, netrep_res$nulls)

mod_by_cond <- mapply(detect_modules, expr_by_cond,
                      lapply(net_by_cond, `[[`, "network"),
                      MoreArgs = list(detailled_result = TRUE),
                      SIMPLIFY = FALSE)

comparison <- compare_conditions(expr_by_cond,
                                 lapply(net_by_cond, `[[`, "adja_mat"),
                                 lapply(net_by_cond, `[[`, "cor_mat"),
                                 lapply(mod_by_cond, `[[`, "modules"),
                                 n_perm = 100)

z_summary(comparison$result$cond1$cond2$observed,
          comparison$result$cond1$cond2$nulls)
```

Index