NetBID2

May 1, 2019

Type Package

Title Network-based Bayesian Inference of Drivers, version 2

Version 0.1.1

Maintainer@R person('Xinran', 'Dong', email = 'xinran.dong@stjude.org')

Description A integrative systems biolgy algorithm to infer driers of phenotype based on data-driven context-specific network and Bayesian inference.

License GPL (>= 2)

Encoding UTF-8

LazyData true

Depends R (>= 3.4.0),
Biobase (>= 2.38.0),
GEOquery (>= 2.46.15),
limma (>= 3.34.9),
DESeq2 (>= 1.20.0),
tximport (>= 1.6.0),
RColorBrewer (>= 1.1-2),
plot3D (>= 1.1.1),
hexbin (>= 1.27.2),
igraph (>= 1.1.2),
plotrix (>= 3.7-3),
biomaRt (>= 2.34.2),
openxlsx (>= 4.1.0),
impute (>= 1.52.0),
msigdbr (>= 6.2.1),
ComplexHeatmap (>= 1.17.1),
umap (>= 0.2.0.0),
plyr (>= 1.8.4),
reshape (>= 0.8.7),
arm (>= 1.10-1),
MCMCglmm (>= 2.26),
ConsensusClusterPlus (>= 1.38.0),
aricode (>= 0.1.1),
GSVA (>= 1.22.4),
ordinal (>= 2019.4-25),
vsn (>= 3.48.1),
rmarkdown (>= 1.11),
kableExtra (>= 1.1.0)

Imports colorspace, rhdf5

Suggests lme4

RoxygenNote 6.1.1

R topics documented:

`bid` *Calculate differential expression (DE)/differential activity (DA) by Bayesian Inference.*

Description

`bid` is a function to get differential expression (DE)/differential activity (DA) by Bayesian Inference.

Usage

```
bid(mat = NULL, use_obs_class = NULL, class_order = NULL,
  class_ordered = TRUE, method = c("MLE", "Bayesian"),
  family = gaussian, pooling = c("full", "no", "partial"),
  prior.V.scale = 0.02, prior.R.nu = 1, prior.G.nu = 2,
  nitt = 13000, burnin = 3000, thin = 10, std = TRUE,
  logTransformed = TRUE, log.base = 2, average.method = "geometric",
  pseudoCount = 0, return_model = FALSE, use_seed = 999,
  verbose = FALSE)
```

Arguments

mat	matrix, input expression/activity matrix for IDs (gene/transcript/probe) belong to one gene/gene set, each column is one sample. The matrix is strongly suggest to contain rownames for IDs and colnames for samples. Supposing geneA has multiple probes A1,A2, in Samples (Case-rep1, Case-rep2, Case-rep3, Control-rep1, Control-rep2, Control-rep3). The mat will be a 2*6 numeric matrix. If the gene only contains one probe, the matrix should be a one-row matrix.
use_obs_class	a vector of characters, the cateogory class for all samples. The order of samples in use_obs_class must be the same with mat if no names of the vector is provided. This vector could be generated by the function get_obs_label to extract this vector from the dataframe of pData(eset) by selecting the column name.
class_order	a vector of characters, the order for the sample classes. Attention: The first class in this order vector will be treated as control. If class_ordered==TRUE, The order must be consistent with the phenotypic trend, such as "low", "medium", "high". Else, only the first order is important. If NULL, will use the alphabetical order in use_obs_class. Default is NULL.
class_ordered	logical, whether the sample class in class_order is ordered or not. Default is TRUE.
method	character, choose from 'MLE' or 'Bayesian'. 'MLE' stands for maximum likelihood estimation, that the function will use generalized linear model(glm/glmer) to fit the data for the expression value and sample phenotype, and use MLE to estimate the regression coefficients. 'Bayesian' means that the function will use Bayesian generalized linear model (bayesglm) or multivariate generalized linear mixed model (MCMCglmm) to fit the data. Default is 'Bayesian'.
family	a description of the error distribution and link function to be used in the model. This can be a character string naming a family function, a family function or the result of a call to a family function. (See family for details of family functions). Currently only support gaussian,poisson,binomial(two group sample class)/category(multi-group sample class)/ordinal(multi-group sample class with class_ordered=TRUE) If set at gaussian or poission, the response variable will be the expression level and the regressors will be the sample phenotype. If set at binomial,the response variable will be the sample phenotype and the regressors will be the expression level. For the input of binomial, category and ordinal, the family will be automatically reset by the input sample class level and the setting of class_ordered. Default is gaussian.
pooling	character, choose from 'full','no','partial'. The strategy for the calculation of DE/DA. Supposing geneA has multiple probes A1,A2, in Samples (Case-rep1, Case-rep2, Case-rep3, Control-rep1, Control-rep2, Control-rep3). The mat contains the expression for probes A1,A2 in all samples. The purpose is to testify the DE of geneA between Case and Control. 'full' means to pull the probe together and treat them as indepedent observations. 'no' means to treat the probe information as an independent variable in the regression model. 'partial' means to treat the probe information as a random effect in the regression model. Default is 'full'.
prior.V.scale	numeric, parameters used in the prior list for MCMCglmm. Useful when setting method as 'Bayesian' and pooling as 'partial'. Default is 0.02
prior.R.nu	numeric, parameters in the prior list for used in MCMCglmm. Useful when setting method as 'Bayesian' and pooling as 'partial'. Default is 1
prior.G.nu	numeric, parameters in the prior list for used in MCMCglmm. Useful when setting method as 'Bayesian' and pooling as 'partial'. Default is 2

nitt	numeric, number of MCMC iterations, parameters used in MCMCglmm. Useful when setting method as 'Bayesian' and pooling as 'partial'. Default is 13000
burnin	numeric, parameters used in MCMCglmm. Useful when setting method as 'Bayesian' and pooling as 'partial'. Default is 3000
thin	numeric, thinning interval, parameters used in MCMCglmm. Useful when setting method as 'Bayesian' and pooling as 'partial'. Default is 10
std	logical, whether to perform std to the original expression matrix. Default is TRUE
logTransformed	logical, whether the original data has been log transformation. Default is TRUE.
log.base	numeric, the base for log transformation, only used when do.logtransform is TRUE. Default is 2.
average.method	character, the strategy to calculate FC (fold change), choose from 'geometric','arithmetic'. Default is 'geometric'.
pseudoCount	integer, pseudo count to add for all value to avoid -Inf in log transformation when calculating FC (fold change).
return_model	logical, indicate what kind of data to return, if TRUE, the regression model will be returned, otherwise the basic statistics will be returned. Default is FALSE.
use_seed	integer, random seed, default is 999.
verbose	logical, whether to print addtional information.Default is FALSE.

Details

It is the inner function for getDE.BID.2G and also could be directly called. More options related with statistics in Bayesian Inference is provided in this function. If user input the ID conversion table use_feature_info, the original input expression matrix could be at probe/transcript level, and the output for DE/DA could be at gene level. Three pooling strategies could be selected for calculation. The P-value is approximately estimated by the posterior distribution of the coefficient and test whether it is significantly different from 0.

Value

one row data.frame containing the output statistics or the regression model if set return_model as TRUE.

Examples

```
mat <- matrix(c(0.50099,1.2108,1.0524,-0.34881,-0.13441,-0.87112,
                1.84579,2.0356,2.6025,1.62954,1.88281,1.29604),
                nrow=2,byrow=TRUE)
rownames(mat) <- c('A1','A2')
colnames(mat) <- c('Case-rep1','Case-rep2','Case-rep3',
                  'Control-rep1','Control-rep2','Control-rep3')
res1 <- bid(mat=mat,
           use_obs_class = c(rep('Case',3),rep('Control',3)),
           class_order = c('Control','Case'))
## Not run:
```

`cal.Activity` *Calculate activity value for all possible drivers*

Description

`cal.Activity` is a function to calculate activity for all possible drivers by input the target list for drivers and the expression matrix for target genes.

Usage

```
cal.Activity(target_list = NULL, cal_mat = NULL, es.method = "mean",
  std = TRUE)
```

Arguments

`target_list`	a list for the target gene information for the drivers. Each object in the list must be a data.frame and should contain one column ("target") to save the target genes. Strongly suggest to follow the NetBID2 pipeline, and the `target_list` could be automatically generated by `get_net2target_list` by running `get.SJAracne.network`.
`cal_mat`	numeric matrix,the expression matrix for all genes/transcripts for calculation.
`es.method`	character, strategy to calculate the activity for the driver, choose from mean, weighted mean, maxmean, absmean; in which the weighted in the weighted mean is the MI (mutual information) value * sign of correlation (use the spearman correlation sign). Default is 'mean'.
`std`	logical, whether to perform std to the original expression matrix. Default is TRUE.

Value

the activity matrix with each row a driver and each column a sample (same sample order with cal_mat)

Examples

```
analysis.par <- list()
analysis.par$out.dir.DATA <- system.file('demo1','driver/DATA/',package = "NetBID2")
NetBID.loadRData(analysis.par=analysis.par,step='ms-tab')
ac_mat <- cal.Activity(target_list=analysis.par$tf.network$target_list,
                       cal_mat=exprs(analysis.par$cal.eset),
                       es.method='weightedmean')
```

`cal.Activity.GS` *Calculate activity value for gene sets.*

Description

`cal.Activity.GS` is a function to calculate activity for all gene sets.

Usage

```
cal.Activity.GS(use_gs2gene = all_gs2gene[c("H", "CP:BIOCARTA",
  "CP:REACTOME", "CP:KEGG")], cal_mat = NULL, es.method = "mean",
  std = TRUE)
```

Arguments

`use_gs2gene`	a list for geneset to genes, the name for the list is the gene set name and the content in each list is the vector for genes belong to that gene set. Default is all_gs2gene[c('H','CP:BIOCARTA','CP:REACTOME','CP:KEGG')] with all_gs2gene loaded by using `gs.preload`.
`cal_mat`	numeric matrix,the expression matrix for all genes/transcripts for calculation.
`es.method`	character, strategy to calculate the activity for the driver, choose from mean, absmean, maxmean, gsva, ssgsea, zscore, plage, the last four options will use `gsva` function. Default is 'mean'.
`std`	logical, whether to perform std to the original expression matrix. Default is TRUE.

Value

the activity matrix with each row a gene set and each column a sample (same sample order with cal_mat)

Examples

```
analysis.par <- list()
analysis.par$out.dir.DATA <- system.file('demo1','driver/DATA/',package = "NetBID2")
NetBID.loadRData(analysis.par=analysis.par,step='ms-tab')
gs.preload(use_spe='Homo sapiens',update=FALSE)
use_gs2gene <- merge_gs(all_gs2gene=all_gs2gene,
                        use_gs=c('H','CP:BIOCARTA','CP:REACTOME','CP:KEGG','C5'))
exp_mat_gene <- exprs(analysis.par$cal.eset)
## each row is a gene symbol, if not, must convert ID first
ac_gs <- cal.Activity.GS(use_gs2gene = use_gs2gene,
                        cal_mat = exp_mat_gene)
```

combineDE *Combine the differential expression (DE)/differential activity (DA) results.*

Description

combineDE is a function aims to combine DE/DA genes/drivers.

Usage

```
combineDE(DE_list, DE_name = NULL, transfer_tab = NULL,
  method = "Stouffer", twosided = TRUE, signed = FALSE)
```

Arguments

DE_list	list of DE results.
DE_name	a vector of DE names, if NULL, will use the names of the DE_list. If not NULL, must match the order of DE_list. Default is NULL.
transfer_tab	data.frame, the transfer table for ID conversion, could be obtained by get_IDtransfer. This transfer table is used for ID mapping for the results of DE_list. The column names must match the DE_name. If NULL, will treat the ID column for each DE results in DE_list as the same type of ID. Default is NULL.
method	character, choose from Stouffer or Fisher, default is "Stouffer".
twosided	logical, whether the pvalues are from two-sided test or not. If not, pvalues must between 0 and 0.5. Default is TRUE.
signed	logical, whether the sign of the pvals will be considered in calculation. Default is TRUE.

Value

a list of DE/DA results, with one more component named "combine" that include the combined results. The original DE/DA results are filtered by the used ID in combination.

Examples

```
analysis.par <- list()
analysis.par$out.dir.DATA <- system.file('demo1','driver/DATA/',package = "NetBID2")
NetBID.loadRData(analysis.par=analysis.par,step='ms-tab')
phe_info <- pData(analysis.par$cal.eset)
each_subtype <- 'G4'
G0 <- rownames(phe_info)[which(phe_info$`subgroup`!=each_subtype)] # get sample list for G0
G1 <- rownames(phe_info)[which(phe_info$`subgroup`==each_subtype)] # get sample list for G1
DE_gene_limma <- getDE.limma.2G(eset=analysis.par$cal.eset,
                                G1=G1,G0=G0,
                                G1_name=each_subtype,
                                G0_name='other')
DA_driver_limma <- getDE.limma.2G(eset=analysis.par$merge.ac.eset,
                                  G1=G1,G0=G0,
                                  G1_name=each_subtype,
                                  G0_name='other')
DE_list <- list(DE=DE_gene_limma,DA=DA_driver_limma)
```

```
g1 <- gsub('(.*)_.*','\\1',DE_list$DA$ID)
transfer_tab <- data.frame(DE=g1,DA=DE_list$DA$ID,stringsAsFactors = FALSE)
res1 <- combineDE(DE_list,transfer_tab=transfer_tab)

## Not run:
each_subtype <- 'G4'
G0 <- rownames(phe_info)[which(phe_info$`subgroup`!=each_subtype)] # get sample list for G0
G1 <- rownames(phe_info)[which(phe_info$`subgroup`==each_subtype)] # get sample list for G1
DE_gene_limma_G4 <- getDE.limma.2G(eset=analysis.par$cal.eset,
                                   G1=G1,G0=G0,
                                   G1_name=each_subtype,
                                   G0_name='other')
each_subtype <- 'SHH'
G0 <- rownames(phe_info)[which(phe_info$`subgroup`!=each_subtype)] # get sample list for G0
G1 <- rownames(phe_info)[which(phe_info$`subgroup`==each_subtype)] # get sample list for G1
DE_gene_limma_SHH <- getDE.limma.2G(eset=analysis.par$cal.eset,
                                    G1=G1,G0=G0,
                                    G1_name=each_subtype,
                                    G0_name='other')
DE_list <- list(G4=DE_gene_limma_G4,SHH=DE_gene_limma_SHH)
res2 <- combineDE(DE_list,transfer_tab=NULL)

## End(Not run)
```

`combinePvalVector` *Combine Pvalues by Fisher or Stouffer's method*

Description

`combinePvalVector` is a function to combine P-values by Stouffer of Fisher method.

Usage

```
combinePvalVector(pvals, method = c("Stouffer", "Fisher"),
  signed = TRUE, twosided = TRUE)
```

Arguments

`pvals,`	a vector of numeric values, the P-value values need to combine. The sign of the P-value will be added to indicate the direction of testing if signed is set to TRUE.
`method`	character, choose from Stouffer or Fisher, default is "Stouffer".
`signed`	logical, whether the sign of the pvals will be considered in calculation. Default is TRUE.
`twosided`	logical, whether the pvalues are from two-sided test or not. If not, pvalues must between 0 and 0.5. Default is TRUE.

Value

a vector contains the 'Z-statistics' and 'P.Value'

Examples

```
combinePvalVector(c(0.1,1e-3,1e-5))
combinePvalVector(c(0.1,1e-3,-1e-5))
```

`db.preload` *Load database files used for NetBID2 into R workspace.*

Description

`db.preload` returns the TF (transcription factors) and Sig (signaling factors) list (tf_sigs) and biomart database information (db_info) by input interested species name and make choice from gene or transcript level.

Usage

```
db.preload(use_level = "transcript", use_spe = "human",
  update = FALSE, TF_list = NULL, SIG_list = NULL,
  input_attr_type = "external_gene_name", main.dir = NULL,
  db.dir = sprintf("%s/db/", main.dir))
```

Arguments

`use_level`	character, either 'transcript' or 'gene', default is 'gene'
`use_spe`	character, input the species name (e.g 'human', 'mouse', 'rat'), default is 'human'
`update`	logical,whether to update if previous RData has been generated, default FALSE
`TF_list`	a character vector,input the list of TF names, if NULL, will use pre-defined list in the package, default NULL
`SIG_list`	a character vector,input the list of SIG names, if NULL, will use pre-defined list in the package, default NULL
`input_attr_type`	character, input the type for the list of TF_list, SIG_list. If no input for TF_list, SIG_list, just leave it to NULL, default NULL. See biomaRt https://bioconductor.org/packages/release/bioc/vignettes/biomaRt/inst/doc/biomaRt.html for more details.
`main.dir`	character, main file path for NetBID2, if NULL, will set to `system.file(package = "NetBID2")`. Default is NULL.
`db.dir`	character, file path for saving the RData, default is db directory under `main.dir` when setting for `main.dir`.

Details

This is a pre-processing function for NetBID2, user could input the species name (e.g human, mouse), analysis level (transcript or gene level) and optionally input TF list or SIG list (otherwise will use list from package data). The function could automatically download information from biomart and save into RData under the db/ directory with specified species name and analysis level.

Value

Reture TRUE if success and FALSE if not. Will load two variables into R workspace, tf_sigs and db_info.

Examples

```
db.preload(use_level='gene',use_spe='human',update=FALSE)

## Not run:
db.preload(use_level='transcript',use_spe='human',update=FALSE)
db.preload(use_level='gene',use_spe='mouse',update=FALSE)

## End(Not run)
```

---

draw.2D *Draw 2D dimension plot for sample cluster visualization.*

Description

draw.2D is a function to draw 2D dimension plot for sample cluster visualization.

Usage

```
draw.2D(X, Y, class_label, xlab = "PC1", ylab = "PC2",
  legend_cex = 0.8, main = "", point_cex = 1)
```

Arguments

X	a vector of numeric values, the first dimension values.
Y	a vector of numeric values, the second dimension values.
class_label	a vector of characters, with the names are the samples (optional) and the values are the sample cluster label. The function will treat that the order are the same for X,Y and class_label.
xlab	character, the label for x-axis.
ylab	character, the label for y-axis.
legend_cex	numeric, the cex for the legend.
main	character, the title for the plot.
point_cex	numeric, the cex for the points.

Value

logical value indicating whether the plot has been successfully generated

Examples

```
mat1 <- matrix(rnorm(2000,mean=0,sd=1),nrow=100,ncol=20)
rownames(mat1) <- paste0('Gene',1:nrow(mat1))
colnames(mat1) <- paste0('Sample',1:ncol(mat1))
pc <- prcomp(t(mat1))$x
pred_label <- kmeans(pc,centers=4)$cluster ## this can use other cluster results
draw.2D(X=pc[,1],Y=pc[,2],class_label=pred_label)
```

draw.2D.ellipse *Draw 2D dimension plot with ellipse for sample cluster visualization.*

Description

draw.2D.ellipse is a function to draw 2D dimension plot with ellipse to cover the samples in the sample cluster for visualization.

Usage

```
draw.2D.ellipse(X, Y, class_label, xlab = "PC1", ylab = "PC2",
  legend_cex = 0.8, main = "", point_cex = 1)
```

Arguments

X	a vector of numeric values, the first dimension values.
Y	a vector of numeric values, the second dimension values.
class_label	a vector of characters, with the names are the samples (optional) and the values are the sample cluster label. The function will treat that the order are the same for X,Y and class_label.
xlab	character, the label for x-axis.
ylab	character, the label for y-axis.
legend_cex	numeric, the cex for the legend.
main	character, the title for the plot.
point_cex	numeric, the cex for the points.

Value

logical value indicating whether the plot has been successfully generated

Examples

```
mat1 <- matrix(rnorm(2000,mean=0,sd=1),nrow=100,ncol=20)
rownames(mat1) <- paste0('Gene',1:nrow(mat1))
colnames(mat1) <- paste0('Sample',1:ncol(mat1))
pc <- prcomp(t(mat1))$x
pred_label <- kmeans(pc,centers=4)$cluster ## this can use other cluster results
draw.2D.ellipse(X=pc[,1],Y=pc[,2],class_label=pred_label)
```

`draw.2D.text` *Draw 2D dimension plot for sample cluster visualization with user-defined text on each point.*

Description

`draw.2D.text` is a function to draw 2D dimension plot for sample cluster visualization.

Usage

```
draw.2D.text(X, Y, class_label, class_text = NULL, xlab = "PC1",
  ylab = "PC2", legend_cex = 0.8, main = "", point_cex = 1,
  text_cex = NULL)
```

Arguments

`X`	a vector of numeric values, the first dimension values.
`Y`	a vector of numeric values, the second dimension values.
`class_label`	a vector of characters, with the names are the samples (optional) and the values are the sample cluster label. The function will treat that the order are the same for X,Y and class_label.
`class_text`	a vector of characters, the user-defined text on each point. If NULL, will use the names of class_label. Default is NULL.
`xlab`	character, the label for x-axis.
`ylab`	character, the label for y-axis.
`legend_cex`	numeric, the cex for the legend.
`main`	character, the title for the plot.
`point_cex`	numeric, the cex for the points.
`text_cex`	numeric, the cex for the points.

Value

logical value indicating whether the plot has been successfully generated

Examples

```
mat1 <- matrix(rnorm(2000,mean=0,sd=1),nrow=100,ncol=20)
rownames(mat1) <- paste0('Gene',1:nrow(mat1))
colnames(mat1) <- paste0('Sample',1:ncol(mat1))
pc <- prcomp(t(mat1))$x
pred_label <- kmeans(pc,centers=4)$cluster ## this can use other cluster results
draw.2D.text(X=pc[,1],Y=pc[,2],class_label=pred_label,
             point_cex=5,text_cex=0.5)
```

draw.3D *Draw 3D dimension plot for sample cluster visualization.*

Description

draw.3D is a function to draw 3D dimension plot for sample cluster visualization.

Usage

```
draw.3D(X, Y, Z, class_label, xlab = "PC1", ylab = "PC2",
  zlab = "PC3", legend_cex = 0.8, main = "", point_cex = 1,
  legend_pos = "topright", legend_ncol = 1, ...)
```

Arguments

X	a vector of numeric values, the first dimension values.
Y	a vector of numeric values, the second dimension values.
Z	a vector of numeric values, the third dimension values.
class_label	a vector of characters, with the names are the samples (optional) and the values are the sample cluster label. The function will treat that the order are the same for X,Y and class_label.
xlab	character, the label for x-axis.
ylab	character, the label for y-axis.
zlab	character, the label for z-axis.
legend_cex	numeric, the cex for the legend.
main	character, the title for the plot.
point_cex	numeric, the cex for the points.
legend_pos	character, the position to put the legend. Default is 'topright'.
legend_ncol	integer, number of columns used to display the legend. Default is 1.
...	other paramters used in scatter3D.

Value

logical value indicating whether the plot has been successfully generated

Examples

```
mat1 <- matrix(rnorm(2000,mean=0,sd=1),nrow=100,ncol=20)
rownames(mat1) <- paste0('Gene',1:nrow(mat1))
colnames(mat1) <- paste0('Sample',1:ncol(mat1))
pc <- prcomp(t(mat1))$x
pred_label <- kmeans(pc,centers=4)$cluster ## this can use other cluster results
draw.3D(X=pc[,1],Y=pc[,2],Z=pc[,3],class_label=pred_label)
```

draw.bubblePlot *Bubble plot for the top drivers in NetBID2 analysis.*

Description

draw.bubblePlot will draw the buble plot for the top drivers and the enriched gene sets for the targets of each driver.

Usage

```
draw.bubblePlot(driver_list = NULL, show_label = driver_list,
  Z_val = NULL, driver_type = NULL, target_list = NULL,
  transfer2symbol2type = NULL, bg_list = NULL, min_gs_size = 5,
  max_gs_size = 500, gs2gene = NULL, use_gs = NULL,
  display_gs_list = NULL, Pv_adj = "none", Pv_thre = 0.1,
  top_geneset_number = 30, top_driver_number = 30, pdf_file = NULL,
  main = "", mark_gene = NULL, driver_cex = 1, gs_cex = 1)
```

Arguments

driver_list	a vector of characters, the name for the top drivers.
show_label	a vector of characters, the name for the top drivers to display on the plot. If NULL, will display the name in driver_list. Default is NULL.
Z_val	a vector of numeric values, the Z statistics for the driver_list. Better to give name to the vector, otherwise will automatically use driver_list as the name.
driver_type	a vector of characters, the bio-type or other characteristics for the driver. If not NULL, will display the type on the plot. Better to give name to the vector, otherwise will automatically use driver_list as the name.
target_list	a list for the target gene information for the drivers. The names for the list must contain the driver in driver_list. Each object in the list must be a data.frame and should contain one column ("target") to save the target genes. Strongly suggest to follow the NetBID2 pipeline, and the target_list could be automatically generated by get_net2target_list by running get.SJAracne.network.
transfer2symbol2type	data.frame, the transfer table for the original ID to the gene symbol and gene biotype (at gene level) or transcript symbol and transcript biotype (at transcript level). strongly suggest to use get_IDtransfer2symbol2type to generate the transfer table.
bg_list	a vector of characters, the background list of genes for analysis. Only accept gene symbols. Default is NULL, will use all genes in the gs2gene as the background list.
min_gs_size	numeric, minimum gene set size for analysis, default is 5.
max_gs_size	numeric, maximum gene set size for analysis, default is 500.
gs2gene	a list for geneset to genes, the name for the list is the gene set name and the content in each list is the vector for genes belong to that gene set. If NULL, will use all_gs2gene loaded by using gs.preload. Default is NULL.

use_gs — a vector of characters, the name for gene set category used for anlaysis. If gs2gene is set to NULL, use_gs must be the subset of `names(all_gs2gene)`. Could check `all_gs2gene_info` for the cateogory description. Default is 'H', 'CP:BIOCARTA', 'CP:REACTOME', 'CP:KEGG'.

display_gs_list — a vector of characters, the list of gene set names to display on the plot. If NULL, the gene sets are shown according to the significant ranking. Default is NULL.

Pv_adj — character, p-value adjustment method, could check `p.adjust.methods` for the available options. Default is 'none'.

Pv_thre — numeric, cutoff value for the adjusted p-values for significance.Default is 0.1.

top_geneset_number — number for the top enriched gene sets to be displayed on the plot. Default is 30.

top_driver_number — number for the top significant drivers to be displayed on the plot. Default is 30.

pdf_file — character, file path for the pdf file to save the figure into pdf format.If NULL, will not generate pdf file. Default is NULL.

main — character, `main` for the title on the plot.

mark_gene — a vector of characters, if not NULL, the drivers in the mark_gene will be labelled red in the draw. Default is NULL.

driver_cex — numeric, `cex` for the driver displayed on the plot. Default is 1.

gs_cex — numeric, `cex` for the gene sets displayed on the plot. Default is 1.

Details

This is a function to draw the bubble plot for the top significant drivers. Each row is a gene set, and each column is a driver. Each bubble represents the enrichment for each driver's target gene in the corresponding gene set. The size for each bubble shows the intersected size for the target gene and the gene set. The color for each bubble shows the significance of enrichment performed by Fisher's Exact Test. Color bar and size bar are shown in the draw. Besides, the target size and the driver gene/transcript bio-type is shown at the bottom of the draw.

Value

Will return logical value indicating whether the plot has been successfully generated

Examples

```
analysis.par <- list()
analysis.par$out.dir.DATA <- system.file('demo1','driver/DATA/',package = "NetBID2")
NetBID.loadRData(analysis.par=analysis.par,step='ms-tab')
ms_tab <- analysis.par$final_ms_tab
sig_driver <- draw.volcanoPlot(dat=ms_tab,label_col='gene_label',
                               logFC_col='logFC.G4.Vs.others_DA',
                               Pv_col='P.Value.G4.Vs.others_DA',
                               logFC_thre=0.4,
                               Pv_thre=1e-7,
                               main='Volcano Plot for G4.Vs.others_DA',
                               show_label=FALSE,
                               label_type = 'origin',
                               label_cex = 0.5)
gs.preload(use_spe='Homo sapiens',update=FALSE)
db.preload(use_level='gene',use_spe='human',update=FALSE)
```

```
use_genes <- unique(analysis.par$merge.network$network_dat$target.symbol)
transfer_tab <- get_IDtransfer2symbol2type(from_type = 'external_gene_name',
                                           use_genes=use_genes,
                                           dataset='hsapiens_gene_ensembl')
## get transfer table !!!
draw.bubblePlot(driver_list=rownames(sig_driver),
               show_label=ms_tab[rownames(sig_driver),'gene_label'],
               Z_val=ms_tab[rownames(sig_driver),'Z.G4.Vs.others_DA'],
               driver_type=ms_tab[rownames(sig_driver),'gene_biotype'],
               target_list=analysis.par$merge.network$target_list,
               transfer2symbol2type=transfer_tab,
               min_gs_size=5,
               max_gs_size=500,use_gs=c('H'),
               top_geneset_number=5,top_driver_number=5,
               main='Bubbleplot for top driver targets',
               gs_cex = 0.4,driver_cex = 0.5)
 ## the cex is set just in case of figure margin too large,
 ## in real case, user could set cex larger or input pdf file name
## Not run:
analysis.par <- list()
analysis.par$out.dir.DATA <- system.file('demo1','driver/DATA/',package = "NetBID2")
NetBID.loadRData(analysis.par=analysis.par,step='ms-tab')
ms_tab <- analysis.par$final_ms_tab
sig_driver <- draw.volcanoPlot(dat=ms_tab,label_col='gene_label',
                               logFC_col='logFC.G4.Vs.others_DA',
                               Pv_col='P.Value.G4.Vs.others_DA',
                               logFC_thre=0.4,
                               Pv_thre=1e-7,
                               main='Volcano Plot for G4.Vs.others_DA',
                               show_label=FALSE,
                               label_type = 'origin',
                               label_cex = 0.5)
gs.preload(use_spe='Homo sapiens',update=FALSE)
use_genes <- unique(analysis.par$merge.network$network_dat$target.symbol)
transfer_tab <- get_IDtransfer2symbol2type(from_type = 'external_gene_name',
                                           use_genes=use_genes,
                                           dataset='hsapiens_gene_ensembl')
## get transfer table !!!
analysis.par$out.dir.PLOT <- getwd() ## directory for saving the pdf files
mark_gene <- c('KCNA1','EOMES','KHDRBS2','RBM24','UNC5D') ## marker for Group4
draw.bubblePlot(driver_list=rownames(sig_driver),
               show_label=ms_tab[rownames(sig_driver),'gene_label'],
               Z_val=ms_tab[rownames(sig_driver),'Z.G4.Vs.others_DA'],
               driver_type=ms_tab[rownames(sig_driver),'gene_biotype'],
               target_list=analysis.par$merge.network$target_list,
               transfer2symbol2type=transfer_tab,
               min_gs_size=5,max_gs_size=500,
               use_gs=use_gs=c('CP:KEGG','CP:BIOCARTA','H'),
               top_geneset_number=30,top_driver_number=50,
               pdf_file = sprintf('%s/bubbledraw.pdf',
               analysis.par$out.dir.PLOT),
               main='Bubbleplot for top driver targets',
               mark_gene=ms_tab[which(ms_tab$geneSymbol %in% mark_gene),
               'originalID_label'])

## End(Not run)
```

`draw.categoryValue` *Box plot and stripchart for the gene's expression levels and driver's activity values in samples with different categories.*

Description

`draw.categoryValue` will draw the box plot with stripchart for the gene's expression levels and/or driver's activity values in samples with different phenotype categories.

Usage

```
draw.categoryValue(ac_val = NULL, exp_val = NULL,
  use_obs_class = NULL, category_color = NULL,
  stripchart_color = get_transparent("black", 0.7), strip_cex = 1,
  class_order = NULL, class_srt = 90, class_cex = 1,
  pdf_file = NULL, main_ac = "", main_exp = "", main_cex = 1)
```

Arguments

`ac_val`	a vector of numeric values, the activity level for the interested driver across all samples.
`exp_val`	a vector of numeric values, the expression level for the interested gene across all samples.
`use_obs_class`	a vector of characters, the cateogory class for all samples. The order of samples in `use_obs_class` must be the same with `ac_val` or `exp_val`. This vector could be generated by the function `get_obs_label` to extract this vector from the dataframe of `pData(eset)` by selecting the column name.
`category_color`	a vector of characters, each item is the color for the class in `class_order`. If NULL, will automatically use function `get.class.color` generate the color bar. Default is NULL.
`stripchart_color`	character, the color for the stripchart. Default is 'black' with transparent alpha set at 0.7.
`strip_cex`	numeric, `cex` for points on the plot. Default is 1.
`class_order`	a vector of characters, the order of category class displayed on the figure. If NULL, will use the alphabetical order of the category class name. Default is NULL.
`class_srt`	numeric, the displayed category class label rotation in degrees. Default is 90.
`class_cex`	numeric, `cex` for the category class label displayed on the plot. Default is 1.
`pdf_file`	character, file path for the pdf file to save the figure into pdf format.If NULL, will not generate pdf file. Default is NULL.
`main_ac`	character, main for the sub-plot of activity level. Default is "".
`main_exp`	character,main for the sub-plot of expression level. Default is "".
`main_cex`	numeric, `cex` for the main title displayed on the plot. Default is 1.

Details

This is a function to draw the gene's expression level and driver's activity values at the same time in one plot across samples with different phenotype categories. Also, only draw of expression level or activity level is accepted.

Value

Will return logical value indicating whether the plot has been successfully generated

Examples

```
analysis.par <- list()
analysis.par$out.dir.DATA <- system.file('demo1','driver/DATA/',package = "NetBID2")
NetBID.loadRDData(analysis.par=analysis.par,step='ms-tab')
ms_tab <- analysis.par$final_ms_tab
sig_driver <- draw.volcanoPlot(dat=ms_tab,label_col='gene_label',
                               logFC_col='logFC.G4.Vs.others_DA',
                               Pv_col='P.Value.G4.Vs.others_DA',
                               logFC_thre=0.4,
                               Pv_thre=1e-7,
                               main='Volcano Plot for G4.Vs.others_DA',
                               show_label=FALSE,
                               label_type = 'origin',
                               label_cex = 0.5)
driver_list <- rownames(sig_driver)
use_driver <- driver_list[3]
exp_mat <- exprs(analysis.par$cal.eset)
## expression,the rownames could match originalID
ac_mat  <- exprs(analysis.par$merge.ac.eset)
## activity,the rownames could match originalID_label
phe_info <- pData(analysis.par$cal.eset)
use_obs_class <- get_obs_label(phe_info = phe_info,'subgroup')
draw.categoryValue(ac_val=ac_mat[use_driver,],
                   exp_val=exp_mat[ms_tab[use_driver,'originalID'],],
                   use_obs_class=use_obs_class,
                   class_order=c('WNT','SHH','G4'),
                   class_srt=30,
                   main_ac = ms_tab[use_driver,'gene_label'],
                   main_exp=ms_tab[use_driver,'geneSymbol'])
## Not run:
analysis.par <- list()
analysis.par$out.dir.DATA <- system.file('demo1','driver/DATA/',package = "NetBID2")
NetBID.loadRDData(analysis.par=analysis.par,step='ms-tab')
ms_tab <- analysis.par$final_ms_tab
sig_driver <- draw.volcanoPlot(dat=ms_tab,label_col='gene_label',
                               logFC_col='logFC.G4.Vs.others_DA',
                               Pv_col='P.Value.G4.Vs.others_DA',
                               logFC_thre=0.4,
                               Pv_thre=1e-7,
                               main='Volcano Plot for G4.Vs.others_DA',
                               show_label=FALSE,
                               label_type = 'origin',
                               label_cex = 0.5)
driver_list <- rownames(sig_driver)
use_driver <- driver_list[3]
exp_mat <- exprs(analysis.par$cal.eset) ## expression,the rownames could match originalID
```

```
ac_mat <- exprs(analysis.par$merge.ac.eset) ## activity,the rownames could match originalID_label
phe_info <- pData(analysis.par$cal.eset)
use_obs_class <- get_obs_label(phe_info = phe_info,'subgroup')
analysis.par$out.dir.PLOT <- getwd() ## directory for saving the pdf files
draw.categoryValue(ac_val=ac_mat[use_driver,],
                   exp_val=exp_mat[ms_tab[use_driver,'originalID'],],
                   use_obs_class=use_obs_class,
                   class_order=c('WNT','SHH','G4'),
                   class_srt=30,
                   main_ac = ms_tab[use_driver,'gene_label'],
                   main_exp=ms_tab[use_driver,'geneSymbol'],
                   pdf_file=sprintf('%s/categoryValue_demo1.pdf',
                   analysis.par$out.dir.PLOT))

## End(Not run)
```

`draw.clustComp` *Draw the cluster comparison between predicted label and observed label.*

Description

`draw.clustComp` is a function to draw the comparison between the predicted label and observed label.

Usage

```
draw.clustComp(pred_label, obs_label, strategy = "ARI", use_col = TRUE,
  low_K = 5, highlight_clust = NULL, main = NULL, clust_cex = 1,
  outlier_cex = 0.3)
```

Arguments

`pred_label`	a vector of characters, the predicted label.
`obs_label`	a vector of characters, the observed label
`strategy`	character, the strategy to compare with labels, choose from 'ARI (adjusted rand index)', 'NMI (normalized mutual information)', 'Jaccard'. Default is 'ARI'.
`use_col`	logical, whether or not to use color in the plot. Default is TRUE.
`low_K`	integer, the lowest number to display on the figures. Number smaller than this will directly display the sample name. Default is 5
`highlight_clust`	a vector of characters, the cluster need to be highlighted on the plot.
`main`	character, the title for the plot.
`clust_cex`	numeric, cex for the cluster label. Default is 1
`outlier_cex`	numeric, cex for the sample names. Default is 0.3

Value

a matrix for the number in the plot

Examples

```
network.par <- list()
network.par$out.dir.DATA <- system.file('demo1','network/DATA/',package = "NetBID2")
NetBID.loadRData(network.par=network.par,step='exp-QC')
mat <- exprs(network.par$net.eset)
phe <- pData(network.par$net.eset)
intgroup <- 'subgroup'
pred_label <- draw.pca.kmeans(mat=mat,all_k = NULL,
                              obs_label=get_obs_label(phe,intgroup),
                              kmeans_strategy='consensus')
draw.clustComp(pred_label,get_obs_label(phe,intgroup),outlier_cex=1,low_K=2,use_col=TRUE)
draw.clustComp(pred_label,get_obs_label(phe,intgroup),outlier_cex=1,low_K=2,use_col=FALSE)
```

---

draw.combineDE *Plot for combined DE (differentiated expressed)/DA (differentiated activity) Vs. original DE/DA*

Description

draw.combineDE draw the image for the combined DE/DA Vs. original DE/DA.

Usage

```
draw.combineDE(DE_list = NULL, main_id = NULL, top_number = 30,
  row_cex = 1, column_cex = 1, text_cex = 1, pdf_file = NULL)
```

Arguments

DE_list	list, a list of DE/DA results, with one more component named "combine" that include the combined results. Strongly suggest to use the output from combineDE.
main_id	character, the main id for display in the figure, must be one of the name in DE_list. If NULL, will use the first name. Default is NULL.
top_number	number for the top significant genes/drivers in the combine results to be displayed on the plot. Default is 30.
row_cex	numeric, cex for the row labels displayed on the plot. Default is 1
column_cex	numeric, cex for the col labels displayed on the plot. Default is 1
text_cex	numeric, cex for the text displayed on the plot. Default is 1
pdf_file	character, file path for the pdf file to save the figure into pdf format.If NULL, will not generate pdf file. Default is NULL.

Details

This plot function need to input the output from combineDE.

Value

logical value indicating whether the plot has been successfully generated

Examples

```
analysis.par <- list()
analysis.par$out.dir.DATA <- system.file('demo1','driver/DATA/',package = "NetBID2")
NetBID.loadRData(analysis.par=analysis.par,step='ms-tab')
phe_info <- pData(analysis.par$cal.eset)
each_subtype <- 'G4'
G0 <- rownames(phe_info)[which(phe_info$`subgroup`!=each_subtype)] # get sample list for G0
G1 <- rownames(phe_info)[which(phe_info$`subgroup`==each_subtype)] # get sample list for G1
DE_gene_limma_G4 <- getDE.limma.2G(eset=analysis.par$cal.eset,
                                   G1=G1,G0=G0,
                                   G1_name=each_subtype,
                                   G0_name='other')
each_subtype <- 'SHH'
G0 <- rownames(phe_info)[which(phe_info$`subgroup`!=each_subtype)] # get sample list for G0
G1 <- rownames(phe_info)[which(phe_info$`subgroup`==each_subtype)] # get sample list for G1
DE_gene_limma_SHH <- getDE.limma.2G(eset=analysis.par$cal.eset,
                                    G1=G1,G0=G0,
                                    G1_name=each_subtype,
                                    G0_name='other')
DE_list <- list(G4=DE_gene_limma_G4,SHH=DE_gene_limma_SHH)
res2 <- combineDE(DE_list,transfer_tab=NULL)
draw.combineDE(res2)
```

---

draw.eset.QC *QC plot for ExpressionSet class object.*

Description

draw.eset.QC is a function to draw the basic QC plots for an ExpressionSet class object. The QC plots include heatmap, pca, density and meansd.

Usage

```
draw.eset.QC(eset, outdir = ".", do.logtransform = FALSE,
  intgroup = NULL, prefix = "", choose_plot = c("heatmap", "pca",
  "density", "meansd"), generate_html = TRUE)
```

Arguments

eset	ExpressionSet class, the input ExpressionSet class object to be plot.
outdir	character, output directory to save the figures. Suggest to set network.par$out.dir.QC or analysis.par$out.dir.QC
do.logtransform	logical, whether to do log transformation before drawing the QC plots. Default is FALSE.
intgroup	a vector of characters, the interested groups from the phenotype information of the eset to be used in plot. If NULL, will automatcially extract all possible groups by get_int_group. Default is NULL.
prefix	character, the prefix for the QC figure name.Default is "".
choose_plot	a vector of characters, choose one or multiple from 'heatmap', 'pca', 'density', 'meansd.' Default is 'heatmap','pca','density','meansd'.

generate_html — logical, whether to generate html file for the report. If TRUE, will generate a html file by rmarkdown, otherwise will generate separate pdf files. Default is TRUE.

Examples

```
## Not run:
network.par <- list()
network.par$out.dir.DATA <- system.file('demo1','network/DATA/',package = "NetBID2")
NetBID.loadRData(network.par=network.par,step='exp-QC')
intgroups <- get_int_group(network.par$net.eset)
network.par$out.dir.QC <- getwd() ## set the output directory
draw.eset.QC(network.par$net.eset,outdir=network.par$out.dir.QC,intgroup=intgroups)

## End(Not run)
```

---

draw.funcEnrich.bar *Bar plot for the result of gene set enrichment analysis.*

Description

draw.funcEnrich.bar will draw the barplot for the result of gene set enrichment analysis.

Usage

```
draw.funcEnrich.bar(funcEnrich_res = NULL, top_number = 30,
  Pv_col = "Ori_P", item_col = "Intersected_items", Pv_thre = 0.1,
  display_genes = FALSE, name_col = "#Name", gs_cex = 0.5,
  gene_cex = 0.5, main = "", bar_col = brewer.pal(8, "RdBu")[7],
  eg_num = 5, pdf_file = NULL)
```

Arguments

funcEnrich_res — data.frame, containing the result for the function enrichment analysis. Prefer the format generated by using funcEnrich.Fisher. If not, users could prepare the required columns and indicate the column names in the following parameters.

top_number — numeric, number for the top enriched gene sets to be displayed on the plot. Default is 30.

Pv_col — character, the name of the column in funcEnrich_res, which contains the P-value. Default is 'Ori_P'.

item_col — character, the name of the column in funcEnrich_res, which contains the detailed intersected gene list, collapsed by ';'. Default is 'Intersected_items'.

Pv_thre — numeric, cutoff value for the p-values. Genes or drivers with lower p-values are remained. Default is 0.1.

display_genes — logical, whether or not to display the intersected genes on the plot. Default is FALSE

name_col — character, the name of the column in funcEnrich_res, which contains the gene set name. Default is '#Name'.

gs_cex — numeric, cex for the gene sets displayed on the plot. Default is 0.5

gene_cex — numeric, cex for the genes displayed on the plot. Default is 0.5

main	character, main for the title on the plot.
bar_col	character, color code for the bar on the plot. Default is brewer.pal(8,'RdBu')[7].
eg_num	numeric, example number of intersected genes shown on the plot. Default is 5.
pdf_file	character, file path for the pdf file to save the figure into pdf format.If NULL, will not generate pdf file. Default is NULL.

Details

This is a function to draw the barplot for the result of gene set enrichment analysis. Two modes, one just display the P-value, and the other could show the top intersected genes for each gene set.

Value

logical value indicating whether the plot has been successfully generated

Examples

```
analysis.par <- list()
analysis.par$out.dir.DATA <- system.file('demo1','driver/DATA/',package = "NetBID2")
NetBID.loadRData(analysis.par=analysis.par,step='ms-tab')
ms_tab <- analysis.par$final_ms_tab
sig_driver <- draw.volcanoPlot(dat=ms_tab,label_col='gene_label',
                               logFC_col='logFC.G4.Vs.others_DA',
                               Pv_col='P.Value.G4.Vs.others_DA',
                               logFC_thre=0.4,
                               Pv_thre=1e-7,
                               main='Volcano Plot for G4.Vs.others_DA',
                               show_label=FALSE,
                               label_type = 'origin',
                               label_cex = 0.5)
gs.preload(use_spe='Homo sapiens',update=FALSE)
res1 <- funcEnrich.Fisher(input_list=ms_tab[rownames(sig_driver),'geneSymbol'],
                          bg_list=ms_tab[,'geneSymbol'],
                          use_gs=c('H','C5'),Pv_thre=0.1,
                          Pv_adj = 'none')
draw.funcEnrich.bar(funcEnrich_res=res1,top_number=5,
                    main='Function Enrichment for Top drivers',
                    gs_cex=0.4,gene_cex=0.5)
draw.funcEnrich.bar(funcEnrich_res=res1,top_number=5,
                    main='Function Enrichment for Top drivers',
                    display_genes = TRUE,eg_num=3,
                    gs_cex=0.6,gene_cex=0.5)
## Not run:
analysis.par <- list()
analysis.par$out.dir.DATA <- system.file('demo1','driver/DATA/',package = "NetBID2")
NetBID.loadRData(analysis.par=analysis.par,step='ms-tab')
ms_tab <- analysis.par$final_ms_tab
sig_driver <- draw.volcanoPlot(dat=ms_tab,label_col='gene_label',
                               logFC_col='logFC.G4.Vs.others_DA',
                               Pv_col='P.Value.G4.Vs.others_DA',
                               logFC_thre=0.4,
                               Pv_thre=1e-7,
                               main='Volcano Plot for G4.Vs.others_DA',
                               show_label=FALSE,
                               label_type = 'origin',
```

```
                                label_cex = 0.5)
gs.preload(use_spe='Homo sapiens',update=FALSE)
res1 <- funcEnrich.Fisher(input_list=ms_tab[rownames(sig_driver),'geneSymbol'],
                          bg_list=ms_tab[,'geneSymbol'],
                          use_gs=c('H','C5'),Pv_thre=0.1,Pv_adj = 'none')
analysis.par$out.dir.PLOT <- getwd() ## directory for saving the pdf files
draw.funcEnrich.bar(funcEnrich_res=res1,top_number=30,
                    main='Function Enrichment for Top drivers',
                    pdf_file=sprintf('%s/funcEnrich_bar_nogene.pdf',
                    analysis.par$out.dir.PLOT))
draw.funcEnrich.bar(funcEnrich_res=res1,top_number=30,
                    main='Function Enrichment for Top drivers',
                    display_genes = TRUE,gs_cex=0.6,
                    pdf_file=sprintf('%s/funcEnrich_bar_withgene.pdf',
                    analysis.par$out.dir.PLOT))

## End(Not run)
```

`draw.funcEnrich.cluster` *Cluster plot for the result of gene set enrichment analysis.*

Description

`draw.funcEnrich.cluster` will draw the cluster for the result of gene set enrichment analysis.

Usage

```
draw.funcEnrich.cluster(funcEnrich_res = NULL, top_number = 30,
  Pv_col = "Ori_P", name_col = "#Name",
  item_col = "Intersected_items", Pv_thre = 0.1, gs_cex = 0.7,
  gene_cex = 0.8, pv_cex = 0.7, main = "", h = 0.95,
  cluster_gs = TRUE, cluster_gene = TRUE, pdf_file = NULL,
  use_genes = NULL, return_mat = FALSE)
```

Arguments

`funcEnrich_res`	data.frame, containing the result for the function enrichment analysis. Prefer the format generated by using `funcEnrich.Fisher`. If not, users could prepare the required columns and indicate the column names in the following parameters.
`top_number`	numeric, number for the top enriched gene sets to be displayed on the plot. Default is 30.
`Pv_col`	character, the name of the column in `funcEnrich_res`, which contains the P-value. Default is 'Ori_P'.
`name_col`	character, the name of the column in `funcEnrich_res`, which contains the gene set name. Default is '#Name'.
`item_col`	character, the name of the column in `funcEnrich_res`, which contains the detailed intersected gene list, collapsed by ';'. Default is 'Intersected_items'.
`Pv_thre`	numeric, cutoff value for the p-values. Genes or drivers with lower p-values are remained. Default is 0.1.
`gs_cex`	numeric, `cex` for the gene sets displayed on the plot. Default is 0.7.

gene_cex	numeric, cex for the genes displayed on the plot. Default is 0.8.
pv_cex	numeric, cex for the P-value displayed on the plot. Default is 0.7.
main	character, main for the title on the plot.
h	numeric, cutoff for the cluster. This parameter will be used in the cutree function. Default is 0.95
cluster_gs	logical, whether or not to cluster gene sets. Default is TRUE.
cluster_gene	logical, whether or not to cluster genes. Default is TRUE.
pdf_file	character, file path for the pdf file to save the figure into pdf format.If NULL, will not generate pdf file. Default is NULL.
use_genes	a vector of characters, gene list used for display in plot, if NULL will display all genes in the top enriched gene sets.Default is NULL.
return_mat	logical, whether or not to return the matrix used for display. Default is FALSE.

Details

This is a function to draw the cluster (genes and gene sets) for the result of gene set enrichment analysis. The cluster is based on the binary matrix representing the gene's existence in the enriched gene sets. Detailed matrix for the cluster, enriched P-value will be displayed on the plot.

Value

if return_mat==FALSE, will return logical value indicating whether the plot has been successfully generated, otherwise will return the matrix used for cluster.

Examples

```
analysis.par <- list()
analysis.par$out.dir.DATA <- system.file('demo1','driver/DATA/',package = "NetBID2")
NetBID.loadRData(analysis.par=analysis.par,step='ms-tab')
ms_tab <- analysis.par$final_ms_tab
sig_driver <- draw.volcanoPlot(dat=ms_tab,label_col='gene_label',
                               logFC_col='logFC.G4.Vs.others_DA',
                               Pv_col='P.Value.G4.Vs.others_DA',
                               logFC_thre=0.4,
                               Pv_thre=1e-7,
                               main='Volcano Plot for G4.Vs.others_DA',
                               show_label=FALSE,
                               label_type = 'origin',
                               label_cex = 0.5)
gs.preload(use_spe='Homo sapiens',update=FALSE)
res1 <- funcEnrich.Fisher(input_list=ms_tab[rownames(sig_driver),'geneSymbol'],
                          bg_list=ms_tab[,'geneSymbol'],
                          use_gs=c('H','C5'),Pv_thre=0.1,Pv_adj = 'none')
draw.funcEnrich.cluster(funcEnrich_res=res1,top_number=30,gs_cex = 0.5,
                        gene_cex=0.9,pv_cex=0.8)
draw.funcEnrich.cluster(funcEnrich_res=res1,top_number=10,gs_cex = 0.6,
                        gene_cex=1,pv_cex=1,
                        cluster_gs=TRUE,cluster_gene = TRUE)
draw.funcEnrich.cluster(funcEnrich_res=res1,top_number=15,gs_cex = 0.8,
                        gene_cex=0.9,pv_cex=0.8,
                        cluster_gs=TRUE,cluster_gene = FALSE)
draw.funcEnrich.cluster(funcEnrich_res=res1,top_number=20,gs_cex = 0.8,
                         gene_cex=0.9,pv_cex=0.8,
```

```
                        cluster_gs=FALSE,cluster_gene = TRUE)
draw.funcEnrich.cluster(funcEnrich_res=res1,top_number=20,gs_cex = 1,
                        gene_cex=1,pv_cex=0.8,
                        cluster_gs=FALSE,cluster_gene = FALSE)
## Not run:
analysis.par <- list()
analysis.par$out.dir.DATA <- system.file('demo1','driver/DATA/',package = "NetBID2")
NetBID.loadRDData(analysis.par=analysis.par,step='ms-tab')
ms_tab <- analysis.par$final_ms_tab
sig_driver <- draw.volcanoPlot(dat=ms_tab,label_col='gene_label',
                               logFC_col='logFC.G4.Vs.others_DA',
                               Pv_col='P.Value.G4.Vs.others_DA',
                               logFC_thre=0.4,
                               Pv_thre=1e-7,
                               main='Volcano Plot for G4.Vs.others_DA',
                               show_label=FALSE,
                               label_type = 'origin',
                               label_cex = 0.5)
gs.preload(use_spe='Homo sapiens',update=FALSE)
res1 <- funcEnrich.Fisher(input_list=ms_tab[rownames(sig_driver),'geneSymbol'],
                          bg_list=ms_tab[,'geneSymbol'],
                          use_gs=c('H','C5'),Pv_thre=0.1,Pv_adj = 'none')
analysis.par$out.dir.PLOT <- getwd() ## directory for saving the pdf files
draw.funcEnrich.cluster(funcEnrich_res=res1,top_number=30,gs_cex = 0.8,
                        gene_cex=0.9,pv_cex=0.8,
                        pdf_file = sprintf('%s/funcEnrich_cluster.pdf',
                        analysis.par$out.dir.PLOT))
draw.funcEnrich.cluster(funcEnrich_res=res1,top_number=30,gs_cex = 1.4,
                        gene_cex=1.5,pv_cex=1.2,
                        pdf_file = sprintf('%s/funcEnrich_clusterBOTH.pdf',
                        analysis.par$out.dir.PLOT),
                        cluster_gs=TRUE,cluster_gene = TRUE)
draw.funcEnrich.cluster(funcEnrich_res=res1,top_number=30,gs_cex = 0.8,
                        gene_cex=0.9,pv_cex=0.8,
                        pdf_file = sprintf('%s/funcEnrich_clusterGS.pdf',
                        analysis.par$out.dir.PLOT),
                        cluster_gs=TRUE,cluster_gene = FALSE)
draw.funcEnrich.cluster(funcEnrich_res=res1,top_number=30,gs_cex = 0.8,
                        gene_cex=0.9,pv_cex=0.8,
                        pdf_file = sprintf('%s/funcEnrich_clusterGENE.pdf',
                        analysis.par$out.dir.PLOT),
                        cluster_gs=FALSE,cluster_gene = TRUE)
draw.funcEnrich.cluster(funcEnrich_res=res1,top_number=30,gs_cex = 1.5,
                        gene_cex=1.4,pv_cex=1.2,
                        pdf_file = sprintf('%s/funcEnrich_clusterNO.pdf',
                        analysis.par$out.dir.PLOT),
                        cluster_gs=FALSE,cluster_gene = FALSE)

## End(Not run)
```

`draw.GSEA` *GSEA (gene set enrichment analysis) plot for a gene set or a driver.*

Description

draw.GSEA will generate a GSEA plot for a gene set (with annotated gene list) or a driver (with target gene list).

Usage

```
draw.GSEA(rank_profile = NULL, use_genes = NULL,
  use_direction = NULL, main = "", pdf_file = NULL,
  annotation = NULL, annotation_cex = 1.2, left_annotation = NULL,
  right_annotation = NULL)
```

Arguments

rank_profile	a vector of numerics. The ranking profile for the differentiated values in a specific sample condition comparison. The names of the vector must be the gene names. The value in the vector could be the logFC or t-statistics.
use_genes	a vector of characters, the list of genes for analysis. The ID must be the subset of the names of rank_profile. This could either be the annotated gene list for the gene set or the target gene list for the driver.
use_direction	a vector of numerics, indicate the direction for the driver and the target gene list. 1 indicates positive regulation and -1 indicates negative regulation. If NULL, will not consider direction information. Default is NULL.
main	character, title for the draw. Default is "".
pdf_file	character, file path for the pdf file to save the figure into pdf format.If NULL, will not generate pdf file. Default is NULL.
annotation	character, annotation for the significance to be displayed on the plot. If NULL, will perform Kolmogorov-Smirnov tests to get significance. Default is NULL.
annotation_cex	numeric, cex for the annotation displayed on the plot. Default is 1.2
left_annotation	character, annotation displayed on the left of the figure representing left condition of the rank_profile. Default is "".
right_annotation	character, annotation displayed on the right of the figure representing right condition of the rank_profile. Default is "".

Details

This is a plot function to draw GSEA for a gene set or a driver by input differentiated expression profile. User could input the annotation text for the significance or the function could display the significance calculated by Kolmogorov-Smirnov tests. ATTENTION: when user input the use_direction, the rank profile will be duplicated (here Zero cross at the middle) and genes with negative direction (-1 in the use_direction list) will be put in the mirror position of the original place.

Value

logical value indicating whether the plot has been successfully generated

Examples

```
analysis.par <- list()
analysis.par$out.dir.DATA <- system.file('demo1','driver/DATA/',package = "NetBID2")
NetBID.loadRDData(analysis.par=analysis.par,step='ms-tab')
ms_tab <- analysis.par$final_ms_tab
sig_driver <- draw.volcanoPlot(dat=ms_tab,label_col='gene_label',
                               logFC_col='logFC.G4.Vs.others_DA',
                               Pv_col='P.Value.G4.Vs.others_DA',
                               logFC_thre=0.4,
                               Pv_thre=1e-7,
                               main='Volcano Plot for G4.Vs.others_DA',
                               show_label=FALSE,
                               label_type = 'origin',
                               label_cex = 0.5)
driver_list <- rownames(sig_driver)
DE_profile <- analysis.par$DE[[1]]$`Z-statistics`;
names(DE_profile) <- rownames(analysis.par$DE[[1]])
use_driver <- driver_list[1]
use_target_genes <- analysis.par$merge.network$target_list[[use_driver]]$target
use_target_direction <- sign(analysis.par$merge.network$target_list[[use_driver]]$spearman) ## 1/-1
annot <- sprintf('P-value: %s',signif(ms_tab[use_driver,'P.Value.G4.Vs.others_DA'],2))

## draw for the driver
draw.GSEA(rank_profile=DE_profile,
          use_genes=use_target_genes,
          use_direction=use_target_direction,
          main=sprintf('GSEA plot for driver %s',
          ms_tab[use_driver,'gene_label']),
          annotation=annot,annotation_cex=1.2,
          left_annotation='high in G4',
          right_annotation='high in others')
draw.GSEA(rank_profile=DE_profile,
          use_genes=use_target_genes,
          use_direction=NULL,
          main=sprintf('GSEA plot for driver %s',
          ms_tab[use_driver,'gene_label']),
          annotation=NULL,annotation_cex=1.2,
          left_annotation='high in G4',
          right_annotation='high in others')

## draw for the gene set
gs.preload(use_spe='Homo sapiens',update=FALSE)
use_target_genes <- all_gs2gene[[1]][[1]]
draw.GSEA(rank_profile=DE_profile,
         use_genes=use_target_genes,
         main=sprintf('GSEA plot for %s',names(all_gs2gene[[1]][1])),
         left_annotation='high in G4',
         right_annotation='high in others')

## Not run:
#' analysis.par <- list()
analysis.par$out.dir.DATA <- system.file('demo1','driver/DATA/',package = "NetBID2")
NetBID.loadRDData(analysis.par=analysis.par,step='ms-tab')
ms_tab <- analysis.par$final_ms_tab
sig_driver <- draw.volcanoPlot(dat=ms_tab,label_col='gene_label',
                               logFC_col='logFC.G4.Vs.others_DA',
```

```
                                Pv_col='P.Value.G4.Vs.others_DA',
                                logFC_thre=0.4,
                                Pv_thre=1e-7,
                                main='Volcano Plot for G4.Vs.others_DA',
                                show_label=FALSE,
                                label_type = 'origin',
                                label_cex = 0.5)
driver_list <- rownames(sig_driver)
DE_profile <- analysis.par$DE[[1]]$`Z-statistics`;
names(DE_profile) <- rownames(analysis.par$DE[[1]])
use_driver <- driver_list[1]
use_target_genes <- analysis.par$merge.network$target_list[[use_driver]]$target
use_target_direction <- sign(analysis.par$merge.network$target_list[[use_driver]]$spearman) ## 1/-1
annot <- sprintf('P-value: %s',signif(ms_tab[use_driver,'P.Value.G4.Vs.others_DA'],2))
analysis.par$out.dir.PLOT <- getwd() ## directory for saving the pdf files
draw.GSEA(rank_profile=DE_profile,use_genes=use_target_genes,
          use_direction=use_target_direction,
          main=sprintf('GSEA plot for driver %s',ms_tab[use_driver,'gene_label']),
          pdf_file = sprintf('%s/GSEA_driver.pdf',
          analysis.par$out.dir.PLOT),
          annotation=annot,annotation_cex=1.2,
          left_annotation='high in G4',
          right_annotation='high in others')

## draw for the gene set
gs.preload(use_spe='Homo sapiens',update=FALSE)
use_target_genes <- all_gs2gene[[1]][[1]]
draw.GSEA(rank_profile=DE_profile,
         use_genes=use_target_genes,
         main=sprintf('GSEA plot for %s',names(all_gs2gene[[1]][1])),
         pdf_file = sprintf('%s/GSEA_GS_each.pdf',analysis.par$out.dir.PLOT),
         left_annotation='high in G4',
         right_annotation='high in others')

## End(Not run)
```

`draw.GSEA.NetBID` *GSEA (gene set enrichment analysis) plot for the list of drivers.*

Description

`draw.GSEA.NetBID` will generate a GSEA plot for the list of drivers, including the target genes' position on the differentiated expression profile, with statistics of differentiated expression (DE) and differentiated activity (DA) for each driver.

Usage

```
draw.GSEA.NetBID(DE = NULL, name_col = NULL, profile_col = NULL,
  profile_trend = "pos2neg", driver_list = NULL,
  show_label = driver_list, driver_DA_Z = NULL, driver_DE_Z = NULL,
  target_list = NULL, top_driver_number = 30, target_nrow = 2,
  target_col = "RdBu", target_col_type = "PN", left_annotation = "",
  right_annotation = "", main = "", profile_sig_thre = 0,
  Z_sig_thre = 1.64, pdf_file = NULL)
```

Arguments

DE	data.frame,the differentiated expression results. This data.frame could be generated by using getDE.limma.2G or getDE.BID.2G. If user want to generate this data.frame by other strategies, the rownames must be the gene names or need one column to be the gene name (set in name_col) and must contain the columns indicating the differentiated expression profile.
name_col	character, the name of the column in DE, which contains the gene name. If NULL, will use the rownames of DE. Default is NULL.
profile_col	character, the name of the column in DE, which will be used as the differentiated expression profile. If DE is created by getDE.limma.2G or getDE.BID.2G, this parameter could be 'logFC' or 't'.
profile_trend	character, the choice of how to display the profile, from high/positive to low/negative ('pos2neg') or low/negative to high/positive ('neg2pos').Default is 'pos2neg'.
driver_list	a vector of characters, the name for the top drivers.
show_label	a vector of characters, the name for the top drivers to display on the plot. If NULL, will display the name in driver_list. Default is NULL.
driver_DA_Z	a vector of numeric values, the Z statistics of differentiated activity (DA) for the driver_list. Better to give name to the vector, otherwise will automatically use driver_list as the name.
driver_DE_Z	a vector of numeric values, the Z statistics of differentiated expression (DE) for the driver_list. Better to give name to the vector, otherwise will automatically use driver_list as the name.
target_list	a list for the target gene information for the drivers. The names for the list must contain the driver in driver_list. Each object in the list must be a data.frame and should contain one column ("target") to save the target genes. Strongly suggest to follow the NetBID2 pipeline, and the target_list could be automatically generated by get_net2target_list by running get.SJAracne.network.
top_driver_number	numeric, number for the top significant drivers to be displayed on the plot. Default is 30.
target_nrow	numeric, number of rows for each driver display on the plot. Two options, 1 or 2. If set to 1, the target genes' position on the profile will be displayed in one row. If set to 2, the target genes' position on the profile will be displayed in two rows, with positive regulated genes displayed on the first row and negative regulated genes displayed on the second row. Default is 2.
target_col	character, choice of color pattern used to display the targets. Two options,'black' or 'RdBu'. If set to 'black', the lines will be colored in black. If set to 'RdBu', the lines will be colored into Red to Blue color bar. If target_col_type is set as 'PN', the positive regulated genes will be colored in red and negative regulated genes in blue. If target_col_type is set as 'DE', the color for the target genes is set according to its value in the differentiated expression profile, with significant high set for red and low for blue. The significant threshold is set by profile_sig_thre. Default is 'RdBu'.
target_col_type	character, choice of the pattern used to display the color for target genes, only work when target_col is set as 'RdBu'. Two options,'PN' or 'DE'. If set as 'PN', the positive regulated genes will be colored in red and negative regulated genes in blue. If set as 'DE', the color for the target genes is set according to its value in the differentiated expression profile, Default is 'PN'.

left_annotation
: character, annotation displayed on the left of the figure representing left condition of the rank_profile. Default is "".

right_annotation
: character, annotation displayed on the right of the figure representing right condition of the rank_profile. Default is "".

main
: character, title for the plot. Default is "".

profile_sig_thre
: numeric, threshold for the absolute values in profile to be treated as significance. Target genes without signifcant values in the profile will be colored in grey. Only work when target_col_type is set as "DE" and target_col is set as "RdBu". Default is 0.

Z_sig_thre
: numeric, threshold for the Z statistics in driver_DA_Z and driver_DE_Z to be treated as signifcance. Only signifcant values will have background color. Default is 1.64.

pdf_file
: character, file path for the pdf file to save the figure into pdf format.If NULL, will not generate pdf file. Default is NULL.

Details

This is a plot function to draw GSEA for the list of drivers by the input of differentiated expression profile. User could choose to display the target genes in one row or two rows, by selecting black color or red to blue color bar.

Value

logical value indicating whether the plot has been successfully generated

Examples

```
analysis.par <- list()
analysis.par$out.dir.DATA <- system.file('demo1','driver/DATA/',package = "NetBID2")
NetBID.loadRDData(analysis.par=analysis.par,step='ms-tab')
ms_tab <- analysis.par$final_ms_tab
comp <- 'G4.Vs.others'
DE <- analysis.par$DE[[comp]]
sig_driver <- draw.volcanoPlot(dat=ms_tab,label_col='gene_label',
                               logFC_col='logFC.G4.Vs.others_DA',
                               Pv_col='P.Value.G4.Vs.others_DA',
                               logFC_thre=0.4,
                               Pv_thre=1e-7,
                               main='Volcano Plot for G4.Vs.others_DA',
                               show_label=FALSE,
                               label_type = 'origin',
                               label_cex = 0.5)
driver_list <- rownames(sig_driver)
draw.GSEA.NetBID(DE=DE,profile_col='t',
                 name_col='ID',
                 profile_trend='neg2pos',
                 driver_list = driver_list,
                 show_label=ms_tab[driver_list,'gene_label'],
                 driver_DA_Z=ms_tab[driver_list,'Z.G4.Vs.others_DA'],
                 driver_DE_Z=ms_tab[driver_list,'Z.G4.Vs.others_DE'],
                 target_list=analysis.par$merge.network$target_list,
```

```
                  top_driver_number=5,
                  target_nrow=2,target_col='RdBu',
                  left_annotation = 'test_left',
                  right_annotation = 'test_right',
                  main='test',target_col_type='DE',
                  Z_sig_thre=1.64,
                  profile_sig_thre = 1.64)
## Not run:
analysis.par <- list()
analysis.par$out.dir.DATA <- system.file('demo1','driver/DATA/',package = "NetBID2")
NetBID.loadRData(analysis.par=analysis.par,step='ms-tab')
ms_tab <- analysis.par$final_ms_tab
comp <- 'G4.Vs.others'
DE <- analysis.par$DE[[comp]]
sig_driver <- draw.volcanoPlot(dat=ms_tab,label_col='gene_label',
                               logFC_col='logFC.G4.Vs.others_DA',
                               Pv_col='P.Value.G4.Vs.others_DA',
                               logFC_thre=0.4,
                               Pv_thre=1e-7,
                               main='Volcano Plot for G4.Vs.others_DA',
                               show_label=FALSE,
                               label_type = 'origin',
                               label_cex = 0.5)
driver_list <- rownames(sig_driver)
analysis.par$out.dir.PLOT <- getwd() ## directory for saving the pdf files
draw.GSEA.NetBID(DE=DE,profile_col='logFC',profile_trend='neg2pos',
                  driver_list = driver_list,
                  show_label=ms_tab[driver_list,'gene_label'],
                  driver_DA_Z=ms_tab[driver_list,'Z.G4.Vs.others_DA'],
                  driver_DE_Z=ms_tab[driver_list,'Z.G4.Vs.others_DE'],
                  target_list=analysis.par$merge.network$target_list,
                  top_driver_number=30,
                  target_nrow=2,
                  target_col='RdBu',
                  left_annotation = 'test_left',
                  right_annotation = 'test_right',
                  main='test',
                  target_col_type='DE',
                  Z_sig_thre=1.64,
                  profile_sig_thre = 1.64,
                  pdf_file=sprintf('%s/NetBID_GSEA_demo1.pdf',
                  analysis.par$out.dir.PLOT))
draw.GSEA.NetBID(DE=DE,profile_col='t',profile_trend='neg2pos',
                  driver_list = driver_list,
                  show_label=ms_tab[driver_list,'gene_label'],
                  driver_DA_Z=ms_tab[driver_list,'Z.G4.Vs.others_DA'],
                  driver_DE_Z=ms_tab[driver_list,'Z.G4.Vs.others_DE'],
                  target_list=analysis.par$merge.network$target_list,
                  top_driver_number=30,
                  target_nrow=1,
                  target_col='RdBu',
                  left_annotation = 'test_left',
                  right_annotation = 'test_right',
                  main='test',target_col_type='PN',
                  Z_sig_thre=1.64,profile_sig_thre = 1.64,
                  pdf_file=sprintf('%s/NetBID_GSEA_demo2.pdf',
                  analysis.par$out.dir.PLOT))
```

```
## End(Not run)
```

---

draw.GSEA.NetBID.GS *GSEA (gene set enrichment analysis) plot for the list of gene sets.*

Description

draw.GSEA.NetBID.GS will generate a GSEA plot for the list of gene sets, including the annotated genes' position on the differentiated expression profile, with statistics of differentiated activity (DA) for each gene set.

Usage

```
draw.GSEA.NetBID.GS(DE = NULL, name_col = NULL, profile_col = NULL,
  profile_trend = "pos2neg", sig_gs_list = NULL, gs_DA_Z = NULL,
  use_gs2gene = NULL, top_gs_number = 30, target_col = "RdBu",
  left_annotation = "", right_annotation = "", main = "",
  profile_sig_thre = 0, Z_sig_thre = 1.64, pdf_file = NULL)
```

Arguments

DE	data.frame,the differentiated expression results. This data.frame could be generated by using getDE.limma.2G or getDE.BID.2G. If user want to generate this data.frame by other strategies, the rownames must be the gene names and must contain the columns indicating the differentiated expression profile.
name_col	character, the name of the column in DE, which contains the gene name. If NULL, will use the rownames of DE. Default is NULL.
profile_col	character, the name of the column in DE, which will be used as the differentiated expression profile. If DE is created by getDE.limma.2G or getDE.BID.2G, this parameter could be 'logFC' or 't'.
profile_trend	character, the choice of how to display the profile, from high/positive to low/negative ('pos2neg') or low/negative to high/positive ('neg2pos').Default is 'pos2neg'.
sig_gs_list	a vector of characters, the name for the top gene sets.
gs_DA_Z	a vector of numeric values, the Z statistics of differentiated activity (DA) for the sig_gs_list. Better to give name to the vector, otherwise will automatically use sig_gs_list as the name.
use_gs2gene	a list for geneset to genes, the name for the list is the gene set name and the content in each list is the vector for genes belong to that gene set. This parameter could be obtained by choosing one from all_gs2gene[['CP:KEGG']], or merge several categories by merge_gs.
top_gs_number	numeric, number for the top significant gene sets to be displayed on the plot. Default is 30.
target_col	character, choice of color pattern used to display the targets. Two options,'black' or 'RdBu'. If set to 'black', the lines will be colored in black. If set to 'RdBu', the lines will be colored into Red to Blue color bar. The color for the annotated genes is set according to its value in the differentiated expression profile, with significant high set for red and low for blue. The significant threshold is set by profile_sig_thre. Default is 'RdBu'.

`left_annotation`
: character, annotation displayed on the left of the figure representing left condition of the rank_profile. Default is "".

`right_annotation`
: character, annotation displayed on the right of the figure representing right condition of the rank_profile. Default is "".

`main`
: character, title for the plot. Default is "".

`profile_sig_thre`
: numeric, threshold for the absolute values in profile to be treated as significance. annotated genes without signifcant values in the profile will be colored in grey. Only work when `target_col_type` is set as "DE" and `target_col` is set as "RdBu". Default is 0.

`Z_sig_thre`
: numeric, threshold for the Z statistics in `driver_DA_Z` and `driver_DE_Z` to be treated as signifcance. Only signifcant values will have background color. Default is 1.64.

`pdf_file`
: character, file path for the pdf file to save the figure into pdf format.If NULL, will not generate pdf file. Default is NULL.

Details

This is a plot function to draw GSEA for the list of gene sets by the input of differentiated expression profile. User could choose to display the annotated genes by selecting black color or red to blue color bar.

Value

logical value indicating whether the plot has been successfully generated

Examples

```
## Not run:
db.preload(use_level='transcript',use_spe='human',update=FALSE)

## get all_gs2gene

analysis.par <- list()
analysis.par$out.dir.DATA <- system.file('demo','gene set/DATA/',package = "NetBID2")
NetBID.loadRData(analysis.par=analysis.par,step='ms-tab')

ms_tab <- analysis.par$final_ms_tab
comp <- 'G4.Vs.others'
DE <- analysis.par$DE[[comp]]
analysis.par$out.dir.PLOT <- getwd()

## directory for saving the pdf files
exp_mat_gene <- exprs(analysis.par$cal.eset)

## calculate activity for all genesets
use_gs2gene <- merge_gs(all_gs2gene=all_gs2gene,
                        use_gs=c('H','CP:BIOCARTA','CP:REACTOME','CP:KEGG','C5'))
ac_gs <- cal.Activity.GS(use_gs2gene = use_gs2gene,cal_mat = exp_mat_gene)

## get DA for the gene set
phe_info <- pData(analysis.par$cal.eset)
```

```
G0  <- rownames(phe_info)[which(phe_info$`subgroup`!='G4')]
# get sample list for G0
G1  <- rownames(phe_info)[which(phe_info$`subgroup`=='G4')]
# get sample list for G1
DA_gs <- getDE.limma.2G(eset=generate.eset(ac_gs),G1=G1,G0=G0,
                        G1_name='G4',G0_name='others')
## or use: DA_gs <- getDE.BID.2G(eset=generate.eset(ac_gs),G1=G1,G0=G0,
                        G1_name='G4',G0_name='others')
## draw vocalno plot for top sig-GS
sig_gs <- draw.volcanoPlot(dat=DA_gs,
                           label_col='ID',
                           logFC_col='logFC',
                           Pv_col='P.Value',
                           logFC_thre=0.25,
                           Pv_thre=1e-4,
                           main='Volcano Plot for gene sets',
                           show_label=TRUE,
                           label_type = 'distribute',
                           label_cex = 0.5,
                           pdf_file=sprintf('%s/vocalno_GS_DA.pdf',
                           analysis.par$out.dir.PLOT))
## GSEA plot for the significant gene sets
draw.GSEA.NetBID.GS(DE=DE,name_col='ID',
                    profile_col='t',profile_trend='pos2neg',
                    sig_gs_list = sig_gs$names,
                    gs_DA_Z=DA_gs[sig_gs$names,'Z-statistics'],
                    use_gs2gene = use_gs2gene,
                    top_gs_number=5,target_col='RdBu',
                    left_annotation = 'test_left',
                    right_annotation = 'test_right',
                    main='test',Z_sig_thre=1.64,profile_sig_thre = 0,
                    pdf_file=sprintf('%s/NetBID_GSEA_GS_demo1.pdf',
                    analysis.par$out.dir.PLOT))

## End(Not run)
```

`draw.heatmap` *Heatmap plot for displaying expression level or activity level for genes and drivers*

Description

`draw.heatmap` draw the heatmap for expression level or activity level for genes and drivers across selected samples.

Usage

```
draw.heatmap(mat = NULL, use_genes = rownames(mat),
  use_gene_label = use_genes, use_samples = colnames(mat),
  use_sample_label = use_samples, phenotype_info = NULL,
  use_phe = NULL, main = "", scale = "none", pdf_file = NULL,
  cluster_rows = TRUE, cluster_columns = TRUE, show_row_names = TRUE,
  show_column_names = TRUE, clustering_distance_rows = "pearson",
  clustering_distance_columns = "pearson", ...)
```

Arguments

mat	numeric matrix, each row as one gene or driver, each column as one sample
use_genes	a vector of characters, the list of genes for display. Default is the rownames(mat).
use_gene_label	a vector of characters, label for the list of genes for display. Default is the use_genes.
use_samples	a vector of characters, the list of samples for display. Default is the colnames(mat).
use_sample_label	a vector of characters, label for the list of samples for display. Default is the use_samples.
phenotype_info	data.frame,contain the sample phenotype information, can be generated by pData(eset). The rownames should match the colnames of mat. Default is NULL.
use_phe	a list of characters, selected phenotype for display,must be the subset of colnames of phenotype_info.Default is NULL.
main	character, title for the draw. Default is "".
scale	character, indicating if the values should be centered and scaled in either the row direction or the column direction, or none. The default is "none".
pdf_file	character, file path for the pdf file to save the figure into pdf format.If NULL, will not generate pdf file. Default is NULL.
cluster_rows, cluster_columns	parameters used in Heatmap, please check for details. Default is TRUE.
show_row_names, show_column_names	parameters used in Heatmap, please check for details. Default is TRUE.
clustering_distance_rows, clustering_distance_columns	parameters used in Heatmap, please check for details. Default is 'pearson'.
...,	more parameters used in Heatmap

Details

This plot function input the expression/activity matrix, with each row as one gene or driver, each column as one sample.

Value

logical value indicating whether the plot has been successfully generated

Examples

```
analysis.par <- list()
analysis.par$out.dir.DATA <- system.file('demo1/driver/DATA/',package = "NetBID2")
NetBID.loadRData(analysis.par=analysis.par,step='ms-tab')
ms_tab <- analysis.par$final_ms_tab
exp_mat <- exprs(analysis.par$cal.eset) ## expression,the rownames matches originalID in ms_tab
ac_mat <- exprs(analysis.par$merge.ac.eset) ## ac,the rownames matches originalID_label in ms_tab
phe_info <- pData(analysis.par$cal.eset) ## phenotype information
sig_driver <- draw.volcanoPlot(dat=ms_tab,label_col='gene_label',
                               logFC_col='logFC.G4.Vs.others_DA',
                               Pv_col='P.Value.G4.Vs.others_DA',
                               logFC_thre=0.4,
                               Pv_thre=1e-7,
                               main='Volcano Plot for G4.Vs.others_DA',
```

```
                                show_label=FALSE,
                                label_type = 'origin',
                                label_cex = 0.5)
sig_driver <- sig_driver[1:10,]
draw.heatmap(mat=exp_mat,use_genes=ms_tab[rownames(sig_driver),'originalID'],
             use_gene_label=ms_tab[rownames(sig_driver),'gene_label'],
             use_samples=colnames(exp_mat),
             use_sample_label=phe_info[colnames(exp_mat),'geo_accession'],
             phenotype_info=phe_info,use_phe=c('gender','pathology','subgroup'),
             main='Expression for Top drivers',scale='row',
             cluster_rows=TRUE,cluster_columns=TRUE,
             clustering_distance_rows='pearson',
             clustering_distance_columns='pearson',
             row_names_gp = gpar(fontsize = 12))
draw.heatmap(mat=ac_mat,use_genes=ms_tab[rownames(sig_driver),'originalID_label'],
              use_gene_label=ms_tab[rownames(sig_driver),'gene_label'],
              use_samples=colnames(exp_mat),
              use_sample_label=phe_info[colnames(exp_mat),'geo_accession'],
              phenotype_info=phe_info,use_phe=c('gender','pathology','subgroup'),
              main='Activity for Top drivers',scale='row',
              cluster_rows=TRUE,cluster_columns=TRUE,
              clustering_distance_rows='pearson',
              clustering_distance_columns='pearson',
              row_names_gp = gpar(fontsize = 6))

## Not run:
analysis.par <- list()
analysis.par$out.dir.DATA <- system.file('demo1/driver/DATA/',package = "NetBID2")
NetBID.loadRData(analysis.par=analysis.par,step='ms-tab')
ms_tab <- analysis.par$final_ms_tab
exp_mat <- exprs(analysis.par$cal.eset) ## expression,the rownames matches originalID in ms_tab
ac_mat <- exprs(analysis.par$merge.ac.eset) ## ac,the rownames matches originalID_label in ms_tab
phe_info <- pData(analysis.par$cal.eset) ## phenotype information
analysis.par$out.dir.PLOT <- getwd() ## directory for saving the pdf files
sig_driver <- draw.volcanoPlot(dat=ms_tab,label_col='gene_label',
                                logFC_col='logFC.G4.Vs.others_DA',
                                Pv_col='P.Value.G4.Vs.others_DA',
                                logFC_thre=0.4,
                                Pv_thre=1e-7,
                                main='Volcano Plot for G4.Vs.others_DA',
                                show_label=FALSE,
                                label_type = 'origin',
                                label_cex = 0.5)
draw.heatmap(mat=exp_mat,use_genes=ms_tab[rownames(sig_driver),'originalID'],
             use_gene_label=ms_tab[rownames(sig_driver),'gene_label'],
             use_samples=colnames(exp_mat),
             use_sample_label=phe_info[colnames(exp_mat),'geo_accession'],
             phenotype_info=phe_info,
             use_phe=c('gender','pathology','subgroup'),
             main='Expression for Top drivers',scale='row',
             cluster_rows=TRUE,cluster_columns=TRUE,
             clustering_distance_rows='pearson',
             clustering_distance_columns='pearson',
             row_names_gp = gpar(fontsize = 12),
             pdf_file=sprintf('%s/heatmap_demo1.pdf',
             analysis.par$out.dir.PLOT))
draw.heatmap(mat=ac_mat,use_genes=ms_tab[rownames(sig_driver),'originalID_label'],
```

```
              use_gene_label=ms_tab[rownames(sig_driver),'gene_label'],
              use_samples=colnames(exp_mat),
              use_sample_label=phe_info[colnames(exp_mat),'geo_accession'],
              phenotype_info=phe_info,
              use_phe=c('gender','pathology','subgroup'),
              main='Activity for Top drivers',scale='row',
              cluster_rows=TRUE,cluster_columns=TRUE,
              clustering_distance_rows='pearson',
              clustering_distance_columns='pearson',
              row_names_gp = gpar(fontsize = 6),
              pdf_file=sprintf('%s/heatmap_demo2.pdf',
              analysis.par$out.dir.PLOT))

## End(Not run)
```

`draw.MICA` *Draw the cluster plot by MICA (cluster algorithm).*

Description

`draw.MICA` is a visualization function to draw the cluster results for the MICA output.

Usage

```
draw.MICA(outdir = NULL, prjname = NULL, all_k = NULL,
  obs_label = NULL, legend_pos = "topleft", legend_cex = 0.8,
  point_cex = 1, plot_type = "2D.ellipse", choose_k_strategy = "ARI",
  visualization_type = "tsne", return_type = "optimal")
```

Arguments

`outdir`	character, the output directory for running MICA.
`prjname`	charater, the project name set for running MICA.
`all_k`	a vector of integers, the pre-defined K to evaluate. If NULL, will use all possible K. Default is NULL.
`obs_label`	a vector of characters, a vector for sample categories with names representing the sample name.
`legend_pos`	character, position for the legend displayed on the plot. Default is 'topleft'.
`legend_cex`	numeric, cex for the legend displayed on the plot. Default is 0.8.
`point_cex`	numeric, cex for the point in the plot. Default is 1.
`plot_type`	character, the type for the plot, choose from '2D' or '2D.ellipse' or '3D'. Default is '2D.ellipse'.
`choose_k_strategy`	character, the strategy to choose the best K, choose from 'ARI (adjusted rand index)', 'NMI (normalized mutual information)', 'Jaccard'. Default is 'ARI'.
`visualization_type`	character,choose from 'tsne', 'umap' or 'mds'. Default is 'tsne'.
`return_type`	character, the strategy to return the results, choose from 'optimal' and 'all'. If choose 'all', cluster results from all k in `all_k` will be returned. Default is 'optimal'.

Details

This function mainly aims for the visulization of sample clusters. The input is the MICA project information. User need to input the real observation label for samples and this function will choose the best K by compared with predicted label and the observed label. The output figure contains two sub-figures, left is labelled by the real observartion label and right is labelled by the predicted label with comparison score (choose from ARI, NMI, Jaccard) shown above. Not suggested when sample size is small.

Value

a vector of predicted label (if `return_type` is 'optimal') and a list of all possible K (if `return_type` is 'all')

`draw.network.QC` *Generate QC plot for the network object*

Description

`draw.network.QC` is a function to draw the QC plot for the network object, mainly the density plot for the degree and the check for scale-free feature.

Usage

```
draw.network.QC(igraph_obj, outdir = NULL, prefix = "",
  directed = TRUE, weighted = FALSE, generate_html = TRUE)
```

Arguments

`igraph_obj`	igraph object, this could be generated by `get.SJAracne.network`
`outdir`	character, the output directory to save the QC figures.
`prefix`	character, the prefix for the QC figure name.Default is "".
`directed`	logical, whether to treat the network as directed network. Default is TRUE.
`weighted`	logical, whether to treat the network as weighted network. Default is FALSE.
`generate_html`	logical, whether to generate html file for the report. If TRUE, will generate a html file by rmarkdown, otherwise will generate separate pdf files. Default is TRUE.

Value

logical value indicating whether the plot has been successfully generated

Examples

```
## Not run:
if(exists('analysis.par')==TRUE) rm(analysis.par)
network.dir <- sprintf('%s/demo1/network/',system.file(package = "NetBID2")) # use demo
network.project.name <- 'project_2019-02-14' # demo project name
project_main_dir <- 'test/'
project_name <- 'test_driver'
analysis.par  <- NetBID.analysis.dir.create(project_main_dir=project_main_dir,
```

```
                                          prject_name=project_name,
                                          network_dir=network.dir,
                                          network_project_name=network.project.name)
analysis.par$tf.network <- get.SJAracne.network(network_file=analysis.par$tf.network.file)
analysis.par$out.dir.PLOT <- getwd() ## directory for saving the pdf files
draw.network.QC(analysis.par$tf.network$igraph_obj,
                outdir=analysis.par$out.dir.QC,prefix='TF_net_')

## End(Not run)
```

`draw.pca.kmeans` *Draw the cluster plot by PCA (visulization algorithm) and Kmeans (cluster algorithm).*

Description

`draw.pca.kmeans` is a visualization function to draw the cluster for the input data matrix.

Usage

```
draw.pca.kmeans(mat = NULL, all_k = NULL, obs_label = NULL,
  legend_pos = "topleft", legend_cex = 0.8, plot_type = "2D.ellipse",
  point_cex = 1, kmeans_strategy = "basic",
  choose_k_strategy = "ARI", return_type = "optimal")
```

Arguments

`mat`	a numeric data matrix, the column (e.g sample) will be clustered by the features (e.g genes) in rows.
`all_k`	a vector of integers, the pre-defined K to evaluate. If NULL, will use all possible K. Default is NULL.
`obs_label`	a vector of characters, a vector for sample categories with names representing the sample name.
`legend_pos`	character, position for the legend displayed on the plot. Default is 'topleft'.
`legend_cex`	numeric, cex for the legend displayed on the plot. Default is 0.8.
`plot_type`	character, the type for the plot, choose from '2D' or '2D.ellipse' or '3D'. Default is '2D.ellipse'.
`point_cex`	numeric, cex for the point in the plot. Default is 1.
`kmeans_strategy`	character, the strategy to run the kmeans algorith, choose from 'basic' and 'consensus', here the consensus kmeans is performed by functions in `ConsensusClusterPlus`. Default is 'basic'.
`choose_k_strategy`	character, the strategy to choose the best K, choose from 'ARI (adjusted rand index)', 'NMI (normalized mutual information)', 'Jaccard'. Default is 'ARI'.
`return_type`	character, the strategy to return the results, choose from 'optimal' and 'all'. If choose 'all', cluster results from all k in `all_k` will be returned. Default is 'optimal'.

Details

This function mainly aims for the visulization of sample clusters. The input is the expression matrix for each row a gene/transcript/probe and each column a sample. User need to input the real observation label for samples and this function will choose the best K by compared with predicted label and the observed label. The output figure contains two sub-figures, left is labelled by the real observartion label and right is labelled by the predicted label with comparison score (choose from ARI, NMI, Jaccard) shown above.

Value

a vector of predicted label (if `return_type` is 'optimal') and a list of all possible K (if `return_type` is 'all')

Examples

```
network.par <- list()
network.par$out.dir.DATA <- system.file('demo1','network/DATA/',package = "NetBID2")
NetBID.loadRData(network.par=network.par,step='exp-QC')
mat <- exprs(network.par$net.eset)
phe <- pData(network.par$net.eset)
intgroup <- get_int_group(network.par$net.eset)
for(i in 1:length(intgroup)){
 print(intgroup[i])
 pred_label <- draw.pca.kmeans(mat=mat,all_k = NULL,obs_label=get_obs_label(phe,intgroup[i]))
 print(table(list(pred_label=pred_label,obs_label=get_obs_label(phe,intgroup[i]))))
}
pred_label <- draw.pca.kmeans(mat=mat,all_k = NULL,
                              obs_label=get_obs_label(phe,intgroup[i]),
                              kmeans_strategy='consensus')
```

`draw.targetNet` *Target network structure plot for the driver.*

Description

`draw.targetNet` will draw the network structure for the selected driver and its target genes.

Usage

```
draw.targetNet(source_label = "", source_z = NULL, edge_score = NULL,
  label_cex = 0.7, source_cex = 1, pdf_file = NULL,
  arrow_direction = "out")
```

Arguments

`source_label`	the label for the driver displayed on the plot.
`source_z`	numeric, the Z statistic for the driver, used to color the driver point. If NULL, the driver will be colored in grey. Default is NULL.
`edge_score`	a vector of numeric values, indicating the correlation between the driver and the target genes. The value ranges from -1 to 1, with positive value indicating postivie regulation and negative value indicating negative correlation. The names for the vector is the gene labels displayed on the plot.

label_cex numeric, cex for the target genes displayed on the plot. Default is 0.7.

source_cex numeric, cex for the source genes displayed on the plot. Default is 1.

pdf_file character, file path for the pdf file to save the figure into pdf format.If NULL, will not generate pdf file. Default is NULL.

arrow_direction character, choose from 'in' or 'out'. Default is 'out'.

Details

This is a function to draw target network structure for the selected driver. The color bar represents the positive (red) or negative (blue) regulation with line width showing the strength.

Value

Will return logical value indicating whether the plot has been successfully generated

Examples

```
source_label <- 'test1'
source_z <- 1.96
edge_score <- (sample(1:200,size=100,replace=TRUE)-100)/100
names(edge_score) <- paste0('G',1:100)
draw.targetNet(source_label=source_label,source_z=source_z,
               edge_score=edge_score)
draw.targetNet(source_label=source_label,source_z=source_z,
               edge_score=edge_score,
               arrow_direction='in',
               source_cex=2)
## Not run:
source_label <- 'test1'
source_z <- 1.96
edge_score <- (sample(1:200,size=100,replace=TRUE)-100)/100
names(edge_score) <- paste0('G',1:100)
analysis.par <- list()
analysis.par$out.dir.PLOT <- getwd()
draw.targetNet(source_label=source_label,source_z=source_z,
               edge_score=edge_score,
               pdf_file=sprintf('%s/targetNet.pdf',
               analysis.par$out.dir.PLOT))

## End(Not run)
```

---

draw.targetNet.TWO *Target network structure plot for two drivers.*

Description

draw.targetNet.TWO will draw the network structure for the selected two drivers and their target genes.

Usage

```
draw.targetNet.TWO(source1_label = "", source2_label = "",
  source1_z = NULL, source2_z = NULL, edge_score1 = NULL,
  edge_score2 = NULL, arrow1_direction = "out",
  arrow2_direction = "out", label_cex = 0.7, source_cex = 1,
  pdf_file = NULL, total_possible_target = NULL, show_test = FALSE)
```

Arguments

source1_label	the label for the first(left) driver displayed on the plot.
source2_label	the label for the second(right) driver displayed on the plot.
source1_z	numeric, the Z statistic for the first driver, used to color the driver point. If NULL, the driver will be colored in grey. Default is NULL.
source2_z	numeric, the Z statistic for the second driver, used to color the driver point. If NULL, the driver will be colored in grey. Default is NULL.
edge_score1	a vector of numeric values, indicating the correlation between the first driver and the target genes. The value ranges from -1 to 1, with positive value indicating postivie regulation and negative value indicating negative correlation. The names for the vector is the gene labels displayed on the plot.
edge_score2	a vector of numeric values, indicating the correlation between the second driver and the target genes. Similar with edge_score1
arrow1_direction	character, the arrow direction for source1, choose from 'in' or 'out'. Default is 'out'.
arrow2_direction	character, the arrow direction for source2, choose from 'in' or 'out'. Default is 'out'.
label_cex	numeric, cex for the target genes displayed on the plot. Default is 0.7.
source_cex	numeric, cex for the source genes displayed on the plot. Default is 1.
pdf_file	character, file path for the pdf file to save the figure into pdf format.If NULL, will not generate pdf file. Default is NULL.
total_possible_target	numeric or a vector of characters. If input numeric, will be the total number of possible targets. If input a vector of characters, will be the background list of all possible target genes. This parameter will be passed to function test.targetNet.overlap to test whether the target genes of the two drivers are significantly intersected. If NULL, will do not perform this test. Default is NULL.
show_test	logical, indicating whether the testing results will be printed and returned. Default is FALSE.

Details

This is a function to draw target network structure for the selected two drivers. The color bar represents the positive (red) or negative (blue) regulation with line width showing the strength.

Value

if show_test==FALSE, will return logical value indicating whether the plot has been successfully generated, otherwise will return the statistics of testing.

Examples

```
source1_label <- 'test1'
source1_z <- 1.96
edge_score1 <- (sample(1:160,size=80,replace=TRUE)-80)/80
names(edge_score1) <- sample(paste0('G',1:1000),size=80)
source2_label <- 'test2'
source2_z <- -2.36
edge_score2 <- (sample(1:240,size=120,replace=TRUE)-120)/120
names(edge_score2) <- sample(paste0('G',1:1000),size=120)
draw.targetNet.TWO(source1_label=source1_label,
               source2_label=source2_label,
               source1_z=source1_z,source2_z=source2_z,
               edge_score1=edge_score1,edge_score2=edge_score2,
               total_possible_target=paste0('G',1:1000),
               show_test=TRUE,label_cex=0.6)

## Not run:
source1_label <- 'test1'
source1_z <- 1.96
edge_score1 <- (sample(1:160,size=100,replace=TRUE)-80)/80
names(edge_score1) <- sample(paste0('G',1:1000),size=80)
source2_label <- 'test2'
source2_z <- -2.36
edge_score2 <- (sample(1:240,size=100,replace=TRUE)-120)/120
names(edge_score2) <- sample(paste0('G',1:1000),size=120)
analysis.par <- list()
analysis.par$out.dir.PLOT <- getwd()
draw.targetNet.TWO(source1_label=source1_label,
               source2_label=source2_label,
               source1_z=source1_z,source2_z=source2_z,
               edge_score1=edge_score1,edge_score2=edge_score2,
               total_possible_target=paste0('G',1:1000),show_test=TRUE,
               pdf_file=sprintf('%s/targetNetTWO.pdf',
               analysis.par$out.dir.PLOT))

## End(Not run)
```

`draw.umap.kmeans` *Draw the cluster plot by UMAP (visulization algorithm) and Kmeans (cluster algorithm).*

Description

`draw.umap.kmeans` is a visualization function to draw the cluster for the input data matrix.

Usage

```
draw.umap.kmeans(mat = NULL, all_k = NULL, obs_label = NULL,
  legend_pos = "topleft", legend_cex = 0.8,
  kmeans_strategy = "basic", choose_k_strategy = "ARI",
  plot_type = "2D.ellipse", point_cex = 1, return_type = "optimal")
```

Arguments

mat	a numeric data matrix, the column (e.g sample) will be clustered by the features (e.g genes) in rows.
all_k	a vector of integers, the pre-defined K to evaluate. If NULL, will use all possible K. Default is NULL.
obs_label	a vector of characters, a vector for sample categories with names representing the sample name.
legend_pos	character, position for the legend displayed on the plot. Default is 'topleft'.
legend_cex	numeric, cex for the legend displayed on the plot. Default is 0.8.
kmeans_strategy	character, the strategy to run the kmeans algorith, choose from 'basic' and 'consensus', here the consensus kmeans is performed by functions in ConsensusClusterPlus. Default is 'basic'.
choose_k_strategy	character, the strategy to choose the best K, choose from 'ARI (adjusted rand index)', 'NMI (normalized mutual information)', 'Jaccard'. Default is 'ARI'.
plot_type	character, the type for the plot, choose from '2D' or '2D.ellipse' or '3D'. Default is '2D.ellipse'.
point_cex	numeric, cex for the point in the plot. Default is 1.
return_type	character, the strategy to return the results, choose from 'optimal' and 'all'. If choose 'all', cluster results from all k in all_k will be returned. Default is 'optimal'.

Details

This function mainly aims for the visulization of sample clusters. The input is the expression matrix for each row a gene/transcript/probe and each column a sample. User need to input the real observation label for samples and this function will choose the best K by compared with predicted label and the observed label. The output figure contains two sub-figures, left is labelled by the real observartion label and right is labelled by the predicted label with comparison score (choose from ARI, NMI, Jaccard) shown above. Not suggested when sample size is small.

Value

a vector of predicted label (if return_type is 'optimal') and a list of all possible K (if return_type is 'all')

Examples

```
network.par <- list()
network.par$out.dir.DATA <- system.file('demo1','network/DATA/',package = "NetBID2")
NetBID.loadRData(network.par=network.par,step='exp-QC')
mat <- exprs(network.par$net.eset)
phe <- pData(network.par$net.eset)
intgroup <- get_int_group(network.par$net.eset)
for(i in 1:length(intgroup)){
 print(intgroup[i])
 pred_label <- draw.umap.kmeans(mat=mat,all_k = NULL,obs_label=get_obs_label(phe,intgroup[i]))
 print(table(list(pred_label=pred_label,obs_label=get_obs_label(phe,intgroup[i]))))
}
pred_label <- draw.umap.kmeans(mat=mat,all_k = NULL,
```

```
                    obs_label=get_obs_label(phe,intgroup[i]),
                    kmeans_strategy='consensus')
```

`draw.volcanoPlot` *Volcano plot for top DE (differentiated expressed) genes or DA (differentiated activity) drivers*

Description

`draw.volcanoPlot` draw the volcano plot for top DE genes or DA drivers, could display the name of the top items in figures and will retrun the list of top items.

Usage

```
draw.volcanoPlot(dat = NULL, label_col = NULL, logFC_col = NULL,
  Pv_col = NULL, logFC_thre = 1.5, Pv_thre = 0.01,
  show_plot = TRUE, xlab = "log2 Fold Change", ylab = "P-value",
  show_label = FALSE, label_cex = 0.5, legend_cex = 0.7,
  label_type = "distribute", main = "", pdf_file = NULL)
```

Arguments

`dat`	data.frame, prefer the formatted master table generated by `generate.masterTable`, if not, must contain the columns for the following required parameters.
`label_col`	character, the name of the column in `dat`, which contains the gene/driver label for display
`logFC_col`	character, the name of the column in `dat`, which contains the logFC value
`Pv_col`	character, the name of the column in `dat`, which contains the P-value
`logFC_thre`	numeric, cutoff value for the logFC. Genes or drivers with absolute logFC value higher than the cutoff are remained.Default is 1.5.
`Pv_thre`	numeric, cutoff value for the p-values. Genes or drivers with lower p-values are remained.Default is 0.01.
`show_plot`	logical, whether or not to draw the figure. Default is TRUE.
`xlab`	character, title for the X axis
`ylab`	character, title for the Y axis
`show_label`	logical, whether or not to display the genes or drivers passed the cutoff on the plot. Default is FALSE
`label_cex`	numeric, `cex` for the label displayed on the plot. Default is 0.5
`legend_cex`	numeric, `cex` for the legend displayed on the plot. Default is 0.7
`label_type`	character, the strategy for label display on the plot, by choosing 'origin' or 'distribute'. Default is 'distribute'.
`main`	character, `main` for the title on the plot.
`pdf_file`	character, file path for the pdf file to save the figure into pdf format.If NULL, will not generate pdf file. Default is NULL.

Details

This plot function input the master table and draw the volcano plot by setting significant threshold for logFC and P-value.

Value

data.frame containing the significant genes or drivers with the following components:

label_col

logFC_col

Pv_col

Examples

```
analysis.par <- list()
analysis.par$out.dir.DATA <- system.file('demo1','driver/DATA/',package = "NetBID2")
NetBID.loadRData(analysis.par=analysis.par,step='ms-tab')
ms_tab <- analysis.par$final_ms_tab
sig_driver <- draw.volcanoPlot(dat=ms_tab,label_col='gene_label',
                               logFC_col='logFC.G4.Vs.others_DA',
                               Pv_col='P.Value.G4.Vs.others_DA',
                               logFC_thre=0.4,
                               Pv_thre=1e-7,
                               main='Volcano Plot for G4.Vs.others_DA',
                               show_label=FALSE,
                               label_type = 'origin',
                               label_cex = 0.5)

## Not run:
analysis.par <- list()
analysis.par$out.dir.DATA <- system.file('demo1','driver/DATA/',package = "NetBID2")
NetBID.loadRData(analysis.par=analysis.par,step='ms-tab')
ms_tab <- analysis.par$final_ms_tab
analysis.par$out.dir.PLOT <- getwd() ## directory for saving the pdf files
sig_driver <- draw.volcanoPlot(dat=ms_tab,label_col='gene_label',
                               logFC_col='logFC.G4.Vs.others_DA',
                               Pv_col='P.Value.G4.Vs.others_DA',
                               logFC_thre=0.4,
                               Pv_thre=1e-7,
                               main='Volcano Plot for G4.Vs.others_DA',
                               show_label=FALSE,
                               label_type = 'origin',
                               label_cex = 0.5)
sig_gene <- draw.volcanoPlot(dat=ms_tab,label_col='geneSymbol',
                               logFC_col='logFC.G4.Vs.others_DE',
                               Pv_col='P.Value.G4.Vs.others_DE',
                               logFC_thre=2,Pv_thre=1e-3,
                               main='Volcano Plot for G4.Vs.others_DE',
                               show_label=FALSE,
                               pdf_file=sprintf('%s/vocalno_nolabel_DE.pdf',
                               analysis.par$out.dir.PLOT))

## End(Not run)
```

find.gsByGene *find.gsByGene*

Description

find.gsByGene

Usage

```
find.gsByGene(gene = NULL, use_gs = NULL, return_type = "name")
```

Arguments

gene	character
use_gs	a vector of characters
return_type	character, all, length, name

funcEnrich.Fisher *Gene set enrichment analysis by Fisher's Exact Test.*

Description

funcEnrich.Fisher will perform gene set enrichment analysis by Fisher's Exact Test.

Usage

```
funcEnrich.Fisher(input_list = NULL, bg_list = NULL, use_gs = NULL,
  gs2gene = NULL, min_gs_size = 5, max_gs_size = 500,
  Pv_adj = "fdr", Pv_thre = 0.1)
```

Arguments

input_list	a vector of characters, the list of genes for analysis. Only accept gene symbols, and gene ID conversion could be done by preparing a transfer table by using get_IDtransfer and using get_name_transfertab to transfer the gene IDs.
bg_list	a vector of characters, the background list of genes for analysis. Only accept gene symbols. Default is NULL, will use all genes in the gs2gene as the background list.
use_gs	a vector of characters, the name for gene set category used for anlaysis. If gs2gene is set to NULL, use_gs must be the subset of names(all_gs2gene). Could check all_gs2gene_info for the cateogory description. If set to 'all', all gene sets in gs2gene will be used. Default is 'H', 'CP:BIOCARTA', 'CP:REACTOME', 'CP:KEGG' if gs2gene is set to NULL (use all_gs2gene). If user input own gs2gene list, use_gs will be set to 'all' as default.
gs2gene	a list for geneset to genes, the name for the list is the gene set name and the content in each list is the vector for genes belong to that gene set. If NULL, will use all_gs2gene loaded by using gs.preload. Default is NULL.
min_gs_size	numeric, minimum gene set size for analysis, default is 5.

max_gs_size	numeric, maximum gene set size for analysis, default is 500.
Pv_adj	character, p-value adjustment method, could check p.adjust.methods for the available options. Default is 'fdr'.
Pv_thre	numeric, cutoff value for the adjusted p-values for significance. Default is 0.1.

Details

This is a function to find significant enriched gene sets for input gene list. Users could prepare gs2gene or use all_gs2gene preloaded by using gs.preload. Background gene list is accepeted.

Value

The function will return a list of gene sets with significant statistics, detailed as follows,

#Name	Name for the enriched gene set
Total_item	Number of background size
Num_item	Number of genes in the gene set (filtered by the background list)
Num_list	Number of input genes for testing (filtered by the background list)
Num_list_item	Number input genes annotated by the gene set (filtered by the background list)
Ori_P	Original P-value from Fisher's Exact Test
Adj_p	Adjusted P-value
Odds_Ratio	Odds ratio by the 2*2 matrix used for Fisher's Exact Test
Intersected_items	List of the intersected genes, collapsed by ';', the number is equal to Num_list_item

Examples

```
analysis.par <- list()
analysis.par$out.dir.DATA <- system.file('demo1','driver/DATA/',package = "NetBID2")
NetBID.loadRData(analysis.par=analysis.par,step='ms-tab')
ms_tab <- analysis.par$final_ms_tab
sig_driver <- draw.volcanoPlot(dat=ms_tab,label_col='gene_label',
                               logFC_col='logFC.G4.Vs.others_DA',
                               Pv_col='P.Value.G4.Vs.others_DA',
                               logFC_thre=0.4,
                               Pv_thre=1e-7,
                               main='Volcano Plot for G4.Vs.others_DA',
                               show_label=FALSE,
                               label_type = 'origin',
                               label_cex = 0.5)
gs.preload(use_spe='Homo sapiens',update=FALSE)
res1 <- funcEnrich.Fisher(input_list=ms_tab[rownames(sig_driver),'geneSymbol'],
                               bg_list=ms_tab[,'geneSymbol'],
                               use_gs=c('H','C5'),
                               Pv_thre=0.1,Pv_adj = 'none')
## Not run:
```

`generate.eset` *Generate expression Set object*

Description

`generate.eset` is a function to generate eSet object by input expression matrix (required), phenotype and feature information (optional).

Usage

```
generate.eset(exp_mat = NULL, phenotype_info = NULL,
  feature_info = NULL, annotation_info = "")
```

Arguments

`exp_mat`	data matrix, the expression data matrix with each row a gene/transcript and each column a sample.
`phenotype_info`	data.frame, the phenotype information for the samples in `exp_mat`, the rownames must match the colnames of `exp_mat`. If NULL, will generate a single column dataframe. Default is NULL.
`feature_info`	data.frame,the feature information for the genes/transcripts/probes in `exp_mat`, the rownames must match the rownames of `exp_mat`. If NULL, will generate a single column dataframe. Default is NULL.
`annotation_info`	character, the annotation for the eSet. Default is "".

Details

This function aims to assist the generation of eSet object, especially for sometimes only expression matrix is available.

Value

an ExressionSet object.

Examples

```
mat1 <- matrix(rnorm(10000),nrow=1000,ncol=10)
colnames(mat1) <- paste0('Sample',1:ncol(mat1))
rownames(mat1) <- paste0('Gene',1:nrow(mat1))
eset <- generate.eset(exp_mat=mat1)
```

`generate.masterTable` *Generate final master table for drivers.*

Description

`generate.masterTable` is a function to automatically generate master tables by input the merged TF/SIG results.

Usage

```
generate.masterTable(use_comp = NULL, DE = NULL, DA = NULL,
  network = NULL, main_id_type = NULL, transfer_tab = NULL,
  tf_sigs = tf_sigs, z_col = "Z-statistics", display_col = c("logFC",
  "P.Value"), column_order_stratey = "type")
```

Arguments

`use_comp`	a vector of characters, the comparison name used to display in the master table.
`DE`	a list of DE results, the list name must contain the items in `use_comp`.
`DA`	a list of DA results, the list name must contain the items in `use_comp`.
`network`	a list for the target gene information for the drivers. Each object in the list must be a data.frame and should contain one column ("target") to save the target genes. Strongly suggest to follow the NetBID2 pipeline, and the `TF_network` could be automatically generated by `get_net2target_list` by running `get.SJAracne.network`.
`main_id_type`	character, the main gene id type. The attribute name from the biomaRt package, such as 'ensembl_gene_id', 'ensembl_gene_id_version', 'ensembl_transcript_id', 'ensembl_transcript_id_version', 'refseq_mrna'. Full list could be obtained by `listAttributes(mart)$name`, where `mart` is the output of `useMart` function.
`transfer_tab`	data.frame, the transfer table for ID conversion, could be obtained by `get_IDtransfer`.
`tf_sigs`	list, which contain the detailed information for the TF and SIGs, this can be obtained by running `db.preload`.
`z_col`	character, column name in `DE`, `DA` that contains the Z statistics. Default is 'Z-statistics'.
`display_col`	character,column name in `DE`, `DA` to be kept in the master table. Default is c('logFC','P.Value').
`column_order_stratey`	character, choose from 'type' and 'comp'. Default is 'type'.

Details

This function is designed for automatically generate master table, with : DE (differentiated expression), DA(differentiated activity for drivers),network (a list for the target gene information for the drivers), and necessary additional information (main_id_type, tf_sigs, z_col and display_col). If results from TF/SIG do not have merged before, please use `generate.masterTable.TF_SIG`.

Value

a data.frame that contains the information for all tested drivers. The column "originalID" and "originalID_label" contain the ID same with the original dataset.

Examples

```
analysis.par <- list()
analysis.par$out.dir.DATA <- system.file('demo1','driver/DATA/',package = "NetBID2")
NetBID.loadRData(analysis.par=analysis.par,step='ms-tab')
#analysis.par$final_ms_tab ## this is master table generated before
ac_mat <- cal.Activity(target_list=analysis.par$merge.network$target_list,
                       cal_mat=exprs(analysis.par$cal.eset),es.method='weightedmean')
analysis.par$ac.merge.eset  <- generate.eset(exp_mat=ac_mat,
                                             phenotype_info=pData(analysis.par$cal.eset))
phe_info <- pData(analysis.par$cal.eset)
all_subgroup <- unique(phe_info$subgroup) ##
for(each_subtype in all_subgroup){
 comp_name <- sprintf('%s.Vs.others',each_subtype) ## each comparison must give a name !!!
 G0  <- rownames(phe_info)[which(phe_info$`subgroup`!=each_subtype)] # get sample list for G0
 G1  <- rownames(phe_info)[which(phe_info$`subgroup`==each_subtype)] # get sample list for G1
 DE_gene_limma <- getDE.limma.2G(eset=analysis.par$cal.eset,G1=G1,G0=G0,
                                 G1_name=each_subtype,G0_name='other')
 analysis.par$DE[[comp_name]] <- DE_gene_limma
 DA_driver_limma <- getDE.limma.2G(eset=analysis.par$ac.merge.eset,G1=G1,G0=G0,
                                   G1_name=each_subtype,G0_name='other')
 analysis.par$DA[[comp_name]] <- DA_driver_limma
}
all_comp <- names(analysis.par$DE) ## get all comparison name for output
db.preload(use_level='gene',use_spe='human',update=FALSE);
test_ms_tab <- generate.masterTable(use_comp=all_comp,
                                    DE=analysis.par$DE,
                                    DA=analysis.par$DA,
                                  network=analysis.par$merge.network$target_list,
                                    tf_sigs=tf_sigs,
                                    z_col='Z-statistics',
                                    display_col=c('logFC','P.Value'),
                                    main_id_type='external_gene_name')
```

---

generate.masterTable.TF_SIG

Generate final master table for drivers.

Description

generate.masterTable.TF_SIG is a function to automatically generate master tables by input TF (transcription factor)/Sig (signaling factor) results separately.

Usage

```
generate.masterTable.TF_SIG(use_comp = NULL, DE = NULL, DA_TF = NULL,
  DA_SIG = NULL, TF_network = NULL, SIG_network = NULL,
  main_id_type = NULL, tf_sigs = tf_sigs, z_col = "Z-statistics",
  display_col = c("logFC", "P.Value"))
```

Arguments

use_comp	a vector of characters, the comparison name used to display in the master table.
DE	a list of DE results, the list name must contain the items in use_comp.

DA_TF	a list of TF DA results, the list name must contain the items in use_comp.
DA_SIG	a list of SIG DA results, the list name must contain the items in use_comp.
TF_network	a list for the target gene information for the TFs. Each object in the list must be a data.frame and should contain one column ("target") to save the target genes. Strongly suggest to follow the NetBID2 pipeline, and the TF_network could be automatically generated by get_net2target_list by running get.SJAracne.network.
SIG_network	a list for the target gene information for the SIGs. Each object in the list must be a data.frame and should contain one column ("target") to save the target genes. Strongly suggest to follow the NetBID2 pipeline, and the TF_network could be automatically generated by get_net2target_list by running get.SJAracne.network.
main_id_type	character, the main gene id type. The attribute name from the biomaRt package, such as 'ensembl_gene_id', 'ensembl_gene_id_version', 'ensembl_transcript_id', 'ensembl_transcript_id_version', 'refseq_mrna'. Full list could be obtained by listAttributes(mart)$name, where mart is the output of useMart function.
tf_sigs	list, which contain the detailed information for the TF and SIGs, this can be obtained by running db.preload.
z_col	character, column name in DE, DA_TF, DA_SIG that contains the Z statistics. Default is 'Z-statistics'.
display_col	character,column name in DE, DA_TF, DA_SIG to be kept in the master table. Default is c('logFC','P.Value').

Details

This function is designed for automatically generate master table, with : DE (differentiated expression), DA_TF (differentiated activity for TF), DA_SIG (differentiated activity for SIG), TF_network (a list for the target gene information for the TF), SIG_network (a list for the target gene information for the SIG), and necessary additional information (main_id_type, tf_sigs, z_col and display_col). If results from TF/SIG have merged before, please use generate.masterTable.

Value

a data.frame that contains the information for all tested drivers The column "originalID" and "originalID_label" contain the ID same with the original dataset.

Examples

```
analysis.par <- list()
analysis.par$out.dir.DATA <- system.file('demo1','driver/DATA/',package = "NetBID2")
NetBID.loadRData(analysis.par=analysis.par,step='ms-tab')
#analysis.par$final_ms_tab ## this is master table generated before
ac_mat <- cal.Activity(target_list=analysis.par$tf.network$target_list,
                       cal_mat=exprs(analysis.par$cal.eset),es.method='weightedmean')
analysis.par$ac.tf.eset  <- generate.eset(exp_mat=ac_mat,
                                          phenotype_info=pData(analysis.par$cal.eset))
ac_mat <- cal.Activity(target_list=analysis.par$sig.network$target_list,
                       cal_mat=exprs(analysis.par$cal.eset),es.method='weightedmean')
analysis.par$ac.sig.eset  <- generate.eset(exp_mat=ac_mat,
                                          phenotype_info=pData(analysis.par$cal.eset))
phe_info <- pData(analysis.par$cal.eset)
all_subgroup <- unique(phe_info$subgroup) ##
for(each_subtype in all_subgroup){
 comp_name <- sprintf('%s.Vs.others',each_subtype) ## each comparison must give a name !!!
```

```
 G0 <- rownames(phe_info)[which(phe_info$`subgroup`!=each_subtype)] # get sample list for G0
 G1 <- rownames(phe_info)[which(phe_info$`subgroup`==each_subtype)] # get sample list for G1
 DE_gene_limma <- getDE.limma.2G(eset=analysis.par$cal.eset,G1=G1,G0=G0,
                                 G1_name=each_subtype,G0_name='other')
 analysis.par$DE[[comp_name]] <- DE_gene_limma
 DA_driver_limma <- getDE.limma.2G(eset=analysis.par$ac.tf.eset,G1=G1,G0=G0,
                                   G1_name=each_subtype,G0_name='other')
 analysis.par$DA_TF[[comp_name]] <- DA_driver_limma
 DA_driver_limma <- getDE.limma.2G(eset=analysis.par$ac.sig.eset,G1=G1,G0=G0,
                                   G1_name=each_subtype,G0_name='other')
 analysis.par$DA_SIG[[comp_name]] <- DA_driver_limma
}
all_comp <- names(analysis.par$DE) ## get all comparison name for output
db.preload(use_level='gene',use_spe='human',update=FALSE);
test_ms_tab <- generate.masterTable.TF_SIG(use_comp=all_comp,
                                           DE=analysis.par$DE,
                                           DA_TF=analysis.par$DA_TF,
                                           DA_SIG=analysis.par$DA_SIG,
                                         TF_network=analysis.par$tf.network$target_list,
                                        SIG_network=analysis.par$sig.network$target_list,
                                           tf_sigs=tf_sigs,
                                           z_col='Z-statistics',
                                           display_col=c('logFC','P.Value'),
                                           main_id_type='external_gene_name')
```

`get.class.color` *Generate a color vector for input character.*

Description

`get.class.color` will generate a vector of colors for input character.

Usage

```
get.class.color(x, use_color = NULL, pre_define = NULL,
  use_inner = TRUE)
```

Arguments

`x`	a vector of characters.
`use_color`	a vector of characters, color bar used to generate the color vector for the input.Default is brewer.pal(12, 'Set3').
`pre_define`	a vector of characters, pre-defined color code for some of the input characters. Default is NULL.
`use_inner`	logical, indicating whether to use inner pre-defined color code for some characters. Default is TRUE.

Details

This is a simple function to generate a vector of colors for input characters. Users could define some of the colors for part of the inputs.

Value

Will return a vector of colors with names the input vector characters.

Examples

```
get.class.color(c('ClassA','ClassB','ClassC','ClassA','ClassC','ClassC'))
get.class.color(c('ClassA','ClassB','ClassC','SHH','WNT','Group3','Group4'),
                use_inner=FALSE)
get.class.color(c('ClassA','ClassB','ClassC','SHH','WNT','Group3','Group4'),
                use_inner=FALSE,use_color=brewer.pal(8, 'Set1'))

pre_define <- c('blue', 'red', 'yellow', 'green','yellow', 'green')
                ## pre-defined colors for MB
names(pre_define) <- c('WNT', 'SHH', 'Group3', 'Group4','GroupC', 'GroupD')
                ##pre-defined color name for MB
get.class.color(c('ClassA','ClassB','ClassC','SHH','WNT','Group3','Group4'),
                pre_define=pre_define,use_inner=FALSE)

## Not run:
```

---

get.SJAracne.network *Generate data structure to save network information by input network file generated by SJAracne*

Description

`get.SJAracne.network` is a function to read in network file generated by SJAracne (consensus_network_ncol_.txt file in the result directory)

Usage

```
get.SJAracne.network(network_file = NULL)
```

Arguments

`network_file`	character, file path for the network file. Must use the file (consensus_network_ncol_.txt) in the result directory. The directory may like: <SJAR_main_dir>/<project_name>/output_[tf|sig]_sja <SJAR_main_dir>/SJARACNE_<project_name>/SJARACNE_out.final consensus_network_ncol_.txt

Details

This function aims to read in network files generated by SJAracne and save the network information into three lists, `network_dat` is a data.frame to save all the information in the network file; `target_list` is a list for containing the target genes' information for the drivers. The names for the list is the driver name and each object in the list is a data.frame to save the target genes. `igraph_obj` is an igraph object to save the network, it is a directed, weighted network. The function will set two edge attributes to the igraph_obj, `weight` is the MI values and `sign` is the sign for the spearman value to indicate positive regulation (1) or negative regulation (-1).

Value

This function will return a list containing three items, network_dat, target_list and igraph_obj.

Examples

```
if(exists('analysis.par')==TRUE) rm(analysis.par)
network.dir <- sprintf('%s/demo1/network/',system.file(package = "NetBID2")) # use demo
network.project.name <- 'project_2019-02-14' # demo project name
project_main_dir <- 'test/'
project_name <- 'test_driver'
analysis.par  <- NetBID.analysis.dir.create(project_main_dir=project_main_dir,
                                            prject_name=project_name,
                                            network_dir=network.dir,
                                            network_project_name=network.project.name)
analysis.par$tf.network  <- get.SJAracne.network(network_file=analysis.par$tf.network.file)
analysis.par$sig.network <- get.SJAracne.network(network_file=analysis.par$sig.network.file)

## Not run:
```

---

get.TF_SIG.list *Get transcription factor (TF) and signaling factor (SIG) list for the input gene/transcript type*

Description

get.TF_SIG.list is a gene ID conversion function to get the TF/SIG list for the input gene list with selected gene/transcript type.

Usage

```
get.TF_SIG.list(use_genes = NULL, use_gene_type = "ensembl_gene_id",
  dataset = NULL)
```

Arguments

use_genes	a vector of characters, all possible genes used in network generation. If NULL, will not filter the TF/SIG list by this gene list. Default is NULL.
use_gene_type	character, attribute name from the biomaRt package, such as 'ensembl_gene_id', 'ensembl_gene_id_version', 'ensembl_transcript_id', 'ensembl_transcript_id_version', 'refseq_mrna'. Full list could be obtained by listAttributes(mart)$name, where mart is the output of useMart function. (e.g mart <- useMart('ensembl',db_info[1])) The type must be the gene type for the input use_genes. Default is 'ensembl_gene_id'.
dataset	character, name for the dataset used for ID conversion, such as 'hsapiens_gene_ensembl'. If NULL, will use db_info[1] if run db.preload brefore. Default is NULL.

Value

This function will return a list containing two parts, for $tf saving the TF list for the input gene type and $sig saving the SIG list.

Examples

```
db.preload(use_level='transcript',use_spe='human',update=FALSE)
use_genes <- c("ENST00000210187","ENST00000216083","ENST00000216127",
               "ENST00000216416","ENST00000217233","ENST00000221418",
               "ENST00000504956","ENST00000507468")
res_list  <- get.TF_SIG.list(use_gene_type = 'ensembl_transcript_id',
                             use_genes=use_genes,
                             dataset='hsapiens_gene_ensembl')
print(res_list)

## Not run:
```

`getDE.BID.2G` *Get differential expression (DE)/differential activity (DA) between case and control sample groups by Bayesian Inference.*

Description

`getDE.BID.2G` is a function aims to get DE/DA genes/drivers with detailed statistical information between case (G1) and control (G0) groups.

Usage

```
getDE.BID.2G(eset, ouput_id_column = NULL, G1 = NULL, G0 = NULL,
  G1_name = NULL, G0_name = NULL, method = "Bayesian",
  family = gaussian, pooling = "full", logTransformed = TRUE,
  verbose = TRUE)
```

Arguments

`eset`	ExpressionSet class object, the input gene expression or driver activity.
`ouput_id_column`	character, the column in the fData(eset) used for output the results. This option is used in the condition that the original expression matrix is at transcript-level, but the statistics can be gene-level if the relation is provided in the feature of the eset. If NULL, will use the rownames of the fData(eset). Default is NULL.
`G1`	a vector of characters, the sample list used as the case.
`G0`	a vecotr of characters, the sample list used as the control.
`G1_name`	character, the group name for the samples in G1, default is "G1".
`G0_name`	character, the group name for the samples in G0, default is "G0".
`method`	character, choose from 'MLE' or 'Bayesian'. 'MLE' stands for maximum likelihood estimation, that the function will use generalized linear model(glm/glmer) to fit the data for the expression value and sample phenotype, and use MLE to estimate the regression coefficients. 'Bayesian' means that the function will use Bayesian generalized linear model (bayesglm) or multivariate generalized linear mixed model (MCMCglmm) to fit the data. Default is 'Bayesian'.

family	a description of the error distribution and link function to be used in the model. This can be a character string naming a family function, a family function or the result of a call to a family function. (See family for details of family functions). Currently only support gaussian,poisson,binomial(two group sample class)/category(multi-group sample class)/ordinal(multi-group sample class with class_ordered=TRUE) If set at gaussian or poission, the response variable will be the expression level and the regressors will be the sample phenotype. If set at binomial,the response variable will be the sample phenotype and the regressors will be the expression level. For the input of binomial, category and ordinal, the family will be automatically reset by the input sample class level and the setting of class_ordered. Default is gaussian.
pooling	character, choose from 'full','no','partial'. The strategy for the calculation of DE/DA. Supposing geneA has multiple probes A1,A2, in Samples (Case-rep1, Case-rep2, Case-rep3, Control-rep1, Control-rep2, Control-rep3). The mat contains the expression for probes A1,A2 in all samples. The purpose is to testify the DE of geneA between Case and Control. 'full' means to pull the probe together and treat them as indepedent observations. 'no' means to treat the probe information as an independent variable in the regression model. 'partial' means to treat the probe information as a random effect in the regression model. Default is 'full'.
logTransformed	logical, whether the original expression value has been log transformed. Default is TRUE.
verbose	logical, whether or not to print verbose information during calculation. Default is TRUE.

Value

A dataframe for all genes with the columns as the output of topTable in limma.

Examples

```
mat <- matrix(c(0.50099,-1.2108,-1.0524,
                0.34881,-0.13441,0.87112,
                1.84579,-2.0356,-2.6025,
                1.62954,1.88281,1.29604),nrow=2,byrow=TRUE)
rownames(mat) <- c('A1','A2')
colnames(mat) <-  c('Case-rep1','Case-rep2','Case-rep3',
                'Control-rep1','Control-rep2','Control-rep3')
tmp_eset <- generate.eset(mat,feature_info=data.frame(row.names=rownames(mat),
            probe=rownames(mat),gene=rep('GeneX',2),
            stringsAsFactors = FALSE))
res1 <- getDE.BID.2G(tmp_eset,ouput_id_column='probe',
        G1=c('Case-rep1','Case-rep2','Case-rep3'),
        G0=c('Control-rep1','Control-rep2','Control-rep3'))
res2 <- getDE.BID.2G(tmp_eset,ouput_id_column='gene',
        G1=c('Case-rep1','Case-rep2','Case-rep3'),
        G0=c('Control-rep1','Control-rep2','Control-rep3'))
res3 <- getDE.BID.2G(tmp_eset,ouput_id_column='gene',
        G1=c('Case-rep1','Case-rep2','Case-rep3'),
        G0=c('Control-rep1','Control-rep2','Control-rep3'),
        pooling='partial')
## Not run:
analysis.par <- list()
analysis.par$out.dir.DATA <- system.file('demo1','driver/DATA/',package = "NetBID2")
```

```
NetBID.loadRData(analysis.par=analysis.par,step='ms-tab')
phe_info <- pData(analysis.par$cal.eset)
each_subtype <- 'G4'
G0 <- rownames(phe_info)[which(phe_info$`subgroup`!=each_subtype)] # get sample list for G0
G1 <- rownames(phe_info)[which(phe_info$`subgroup`==each_subtype)] # get sample list for G1
DE_gene_BID <- getDE.BID.2G(eset=analysis.par$cal.eset,
                                G1=G1,G0=G0,
                                G1_name=each_subtype,
                                G0_name='other')
DA_driver_BID <- getDE.BID.2G(eset=analysis.par$merge.ac.eset,
                                G1=G1,G0=G0,
                                G1_name=each_subtype,
                                G0_name='other')

## End(Not run)
```

`getDE.limma.2G` *Get differential expression (DE)/differential activity (DA) between case and control sample groups by limma related functions.*

Description

`getDE.limma.2G` is a function aims to get DE/DA genes/drivers with detailed statistical information between case (G1) and control (G0) groups.

Usage

```
getDE.limma.2G(eset = NULL, G1 = NULL, G0 = NULL, G1_name = NULL,
  G0_name = NULL, verbose = TRUE, random_effect = NULL)
```

Arguments

`eset`	ExpressionSet class object, the input gene expression or driver activity.
`G1`	a vector of characters, the sample list used as the case.
`G0`	a vecotr of characters, the sample list used as the control.
`G1_name`	character, the group name for the samples in G1, default is "G1".
`G0_name`	character, the group name for the samples in G0, default is "G0".
`verbose`	logical, whether or not to print verbose information during calculation.Default is TRUE.
`random_effect`	a vector of characters, vector or factor specifying a blocking variable. Default is NULL indicating no random effect will be considered.

Value

A dataframe for all genes with the columns as the output of `topTable` in limma.

Examples

```
analysis.par <- list()
analysis.par$out.dir.DATA <- system.file('demo1','driver/DATA/',package = "NetBID2")
NetBID.loadRData(analysis.par=analysis.par,step='ms-tab')
phe_info <- pData(analysis.par$cal.eset)
each_subtype <- 'G4'
G0 <- rownames(phe_info)[which(phe_info$`subgroup`!=each_subtype)] # get sample list for G0
G1 <- rownames(phe_info)[which(phe_info$`subgroup`==each_subtype)] # get sample list for G1
DE_gene_limma <- getDE.limma.2G(eset=analysis.par$cal.eset,
                                G1=G1,G0=G0,
                                G1_name=each_subtype,
                                G0_name='other')
DA_driver_limma <- getDE.limma.2G(eset=analysis.par$merge.ac.eset,
                                G1=G1,G0=G0,
                                G1_name=each_subtype,
                                G0_name='other')
```

---

get_clustComp *Get Score between predicted label and observed label.*

Description

get_clustComp is a function to compare the predicted label and observed label and return the score.

Usage

```
get_clustComp(pred_label, obs_label, strategy = "ARI")
```

Arguments

pred_label	a vector of characters, the predicted label.
obs_label	a vector of characters, the observed label
strategy	character, the strategy to compare with labels, choose from 'ARI (adjusted rand index)', 'NMI (normalized mutual information)', 'Jaccard'. Default is 'ARI'.

Value

score for the comparison

Examples

```
obs_label <- c('A','A','A','B','B','C','D')
pred_label  <- c(1,1,1,1,2,2,2)
get_clustComp(pred_label,obs_label)
```

get_IDtransfer *Gene ID conversion related functions.*

Description

get_IDtransfer will generate a transfer table for ID conversion by input the from-gene type and to-gene type.

Usage

```
get_IDtransfer(from_type = NULL, to_type = NULL, add_type = NULL,
  use_genes = NULL, dataset = NULL, ignore_version = FALSE)
```

Arguments

from_type	character, attribute name from the biomaRt package, such as 'ensembl_gene_id', 'ensembl_gene_id_version', 'ensembl_transcript_id', 'ensembl_transcript_id_version', 'refseq_mrna'. Full list could be obtained by listAttributes(mart)$name, where mart is the output of useMart function. The type must be the gene type for the input use_genes.
to_type	character, character, attribute name from the biomaRt package, the gene type will be convereted to.
add_type	character, character, attribute name from the biomaRt package, additional type in the final table
use_genes	a vector of characters, gene list used for ID conversion. If NULL, will extract all possible genes.
dataset	character, name for the dataset used for ID conversion, such as 'hsapiens_gene_ensembl'. If NULL, will use db_info[1] if run db.preload brefore. Default is NULL.
ignore_version	logical, whether to ignore version when from_type is 'ensembl_gene_id_version' or 'ensembl_transcript_id_version'. Default is FALSE.

Value

get_IDtransfer will return a data.frame for the transfer table.

Examples

```
use_genes <- c("ENST00000210187","ENST00000216083","ENST00000216127",
             "ENST00000216416","ENST00000217233","ENST00000221418")
transfer_tab <- get_IDtransfer(from_type = 'ensembl_transcript_id',
                               to_type='external_gene_name',
                               use_genes=use_genes,
                               dataset='hsapiens_gene_ensembl')
                               ## get transfer table !!!
res1 <- get_name_transfertab(use_genes,transfer_tab=transfer_tab)
transfer_tab_withtype <- get_IDtransfer2symbol2type(from_type = 'ensembl_transcript_id',
                                                    use_genes=use_genes,
                                                    dataset='hsapiens_gene_ensembl')
                                                    ## get transfer table !!!
## Not run:
```

get_IDtransfer2symbol2type
Gene ID conversion related functions.

Description

get_IDtransfer2symbol2type will generate the transfer table for the original ID to the gene symbol and gene biotype (at gene level) or transcript symbol and transcript biotype (at transcript level).

Usage

```
get_IDtransfer2symbol2type(from_type = NULL, use_genes = NULL,
  dataset = NULL, use_level = "gene", ignore_version = FALSE)
```

Arguments

from_type	character, attribute name from the biomaRt package, such as 'ensembl_gene_id', 'ensembl_gene_id_version', 'ensembl_transcript_id', 'ensembl_transcript_id_version', 'refseq_mrna'. Full list could be obtained by listAttributes(mart)$name, where mart is the output of useMart function. The type must be the gene type for the input use_genes.
use_genes	a vector of characters, gene list used for ID conversion. If NULL, will output all possible genes in transfer table. Default is NULL.
dataset	character, name for the dataset used for ID conversion, such as 'hsapiens_gene_ensembl'. If NULL, will use db_info[1] if run db.preload brefore. Default is NULL.
use_level	character, either 'transcript' or 'gene', default is 'gene'
ignore_version	logical, whether to ignore version when from_type is 'ensembl_gene_id_version' or 'ensembl_transcript_id_version'. Default is FALSE.

Value

get_IDtransfer2symbol2type will return a data.frame for the transfer table with gene/transcript biotype.

Examples

```
use_genes <- c("ENST00000210187","ENST00000216083","ENST00000216127",
             "ENST00000216416","ENST00000217233","ENST00000221418")
transfer_tab <- get_IDtransfer(from_type = 'ensembl_transcript_id',
                               to_type='external_gene_name',use_genes=use_genes,
                               dataset='hsapiens_gene_ensembl')
                               ## get transfer table !!!
res1 <- get_name_transfertab(use_genes,transfer_tab=transfer_tab)
transfer_tab_withtype <- get_IDtransfer2symbol2type(from_type = 'ensembl_transcript_id',
                                                    use_genes=use_genes,
                                                    dataset='hsapiens_gene_ensembl',
                                                    use_level='transcript')
                                                    ## get transfer table !!!
## Not run:
```

get_IDtransfer_betweenSpecies
Gene ID conversion related functions.

Description

get_IDtransfer_betweenSpecies will generate a transfer table for ID conversion between species.

Usage

```
get_IDtransfer_betweenSpecies(from_spe = "human", to_spe = "mouse",
  from_type = NULL, to_type = NULL, use_genes = NULL)
```

Arguments

from_spe	character, input the species name (e.g 'human', 'mouse', 'rat') that the use_genes belong to, default is 'human'
to_spe	character, input the species name (e.g 'human', 'mouse', 'rat') that need to transfered to, default is 'mouse'
from_type	character, attribute name from the biomaRt package, such as 'ensembl_gene_id', 'ensembl_gene_id_version' 'ensembl_transcript_id', 'ensembl_transcript_id_version', 'refseq_mrna'. Full list could be obtained by listAttributes(mart)$name, where mart is the output of useMart function. The type must be the gene type for the input use_genes.
to_type	character, character, attribute name from the biomaRt package, the gene type will be convereted to.
use_genes	a vector of characters, gene list used for ID conversion. Must be the genes with from_type in from_spe. If NULL, will output all possible genes in transfer table. Default is NULL.

Value

get_IDtransfer_betweenSpecies will return a data.frame for the transfer table.

Examples

```
use_genes <- c("ENST00000210187","ENST00000216083","ENST00000216127",
              "ENST00000216416","ENST00000217233","ENST00000221418")
transfer_tab <- get_IDtransfer_betweenSpecies(from_spe='human',
                                to_spe='mouse',
                                from_type = 'ensembl_transcript_id',
                                to_type='external_gene_name',
                                use_genes=use_genes)
                                ## get transfer table !!!
transfer_tab <- get_IDtransfer_betweenSpecies(from_spe='human',
                                to_spe='mouse',
                                from_type = 'ensembl_transcript_id',
                                to_type='ensembl_transcript_id_version',
                                use_genes=use_genes)
                                ## get transfer table !!!
## Not run:
```

```
transfer_tab <- get_IDtransfer_betweenSpecies(from_spe='human',
                                to_spe='mouse',
                                from_type='refseq_mrna',
                                to_type='refseq_mrna')

## End(Not run)
```

---

get_int_group *Get interested groups from the pData of an ExpressionSet object.*

Description

get_int_group is a simple function to extract columns with unique sample feature ranges from 2 to sample size-1.

Usage

```
get_int_group(eset)
```

Arguments

eset,	an ExpressionSet object, the input object for analysis.

Value

a vector of characters, the column names which could be used for sample cluster analysis

Examples

```
network.par <- list()
network.par$out.dir.DATA <- system.file('demo1','network/DATA/',package = "NetBID2")
NetBID.loadRDData(network.par=network.par,step='exp-QC')
intgroups <- get_int_group(network.par$net.eset)
```

---

get_name_transfertab *Gene ID conversion related functions.*

Description

get_name_transfertab will get the transfered ID by input the original ID and transfer table.

Usage

```
get_name_transfertab(use_genes = NULL, transfer_tab = NULL,
  from_type = NULL, to_type = NULL, ignore_version = FALSE)
```

Arguments

use_genes	a vector of characters, gene list used for ID conversion.
transfer_tab	data.frame, the transfer table for ID conversion, could be obtained by get_IDtransfer.
from_type	character, attribute name from the biomaRt package, such as 'ensembl_gene_id', 'ensembl_gene_id_version' 'ensembl_transcript_id', 'ensembl_transcript_id_version', 'refseq_mrna'. Full list could be obtained by listAttributes(mart)$name, where mart is the output of useMart function. The type must be the gene type for the input use_genes. If NULL, will use the first column in the transfer table.
to_type	character, character, attribute name from the biomaRt package, the gene type will be convereted to. If NULL, will use the second column in the transfer table.
ignore_version	logical, whether to ignore version when from_type is 'ensembl_gene_id_version' or 'ensembl_transcript_id_version'. Default is FALSE.

Value

get_name_transfertab will return the list for the converted IDs.

Examples

```
use_genes <- c("ENST00000210187","ENST00000216083","ENST00000216127",
             "ENST00000216416","ENST00000217233","ENST00000221418")
transfer_tab <- get_IDtransfer(from_type = 'ensembl_transcript_id',
                               to_type='external_gene_name',use_genes=use_genes,
                               dataset='hsapiens_gene_ensembl')
                               ## get transfer table !!!
res1 <- get_name_transfertab(use_genes=use_genes,transfer_tab=transfer_tab)
transfer_tab_withtype <- get_IDtransfer2symbol2type(from_type = 'ensembl_transcript_id',
                                                    use_genes=use_genes,
                                                    dataset='hsapiens_gene_ensembl')
                                                    ## get transfer table !!!
## Not run:
```

---

get_net2target_list *Get target list by input network information from data.frame.*

Description

get_net2target_list is included in the get.SJAracne.network, the reason to make it invokable just for user to read in network files prepared by themselves.

Usage

```
get_net2target_list(net_dat = NULL)
```

Arguments

net_dat	data.frame, must contain columns named with "source" and "target", "MI" and "spearman" are strongly suggested but not required. If these two columns missed, some options may be error in some of the functions in the following steps (such as es.method='weightedmean' in cal.Activity).

Value

a list for the target gene information for the drivers. Each object in the list is a data.frame to save the target genes.

Examples

```
tf.network.file <- sprintf('%s/demo1/network/SJAR/project_2019-02-14/%s/%s',
                   system.file(package = "NetBID2"),
                   'output_tf_sjaracne_project_2019-02-14_out_.final',
                   'consensus_network_ncol_.txt')
net_dat      <- read.delim(file=tf.network.file,stringsAsFactors = FALSE)
target_list  <- get_net2target_list(net_dat)
```

---

get_obs_label *Generate a vector for sample category.*

Description

get_obs_label will generate a vector for sample categories with names to the vector representing the sample name.

Usage

```
get_obs_label(phe_info, use_col, collapse = "|")
```

Arguments

phe_info	data.frame, phenotype dataframe for the samples with sample names in rownames, e.g from pData(eset).
use_col	a vector of numeric or character, the column index or column name for extraction to get the sample category vector.
collapse	character, character string to separate the results when the length use_col is more than 1. default is "|".

Details

This is a simple function to generate sample category vector from a data.frame. Mainly used for input preparation in the visualization plots.

Value

Will return a vector for sample categories with names to the vector representing the sample name.

Examples

```
analysis.par <- list()
analysis.par$out.dir.DATA <- system.file('demo1','driver/DATA/',package = "NetBID2")
NetBID.loadRData(analysis.par=analysis.par,step='ms-tab')
phe_info <- pData(analysis.par$cal.eset)
use_obs_class <- get_obs_label(phe_info = phe_info,'subgroup')
print(use_obs_class)
## Not run:
```

gs.preload *Load MSigDB for NetBID2 into R workspace.*

Description

gs.preload will load two variables into R workspace, the list for gene set to genes (all_gs2gene) and a dataframe (all_gs2gene_info) for detailed description for gene sets.

Usage

```
gs.preload(use_spe = "Homo sapiens", update = FALSE, main.dir = NULL,
  db.dir = sprintf("%s/db/", main.dir))
```

Arguments

use_spe	character, input the species name (e.g 'Homo sapiens', 'Mus musculus'). Full list of available species name could be found by msigdbr_show_species(). Default is 'Homo sapiens'
update	logical,whether to update if previous RData has been generated, default FALSE
main.dir	character, main file path for NetBID2, if NULL, will set to system.file(package = "NetBID2"). Default is NULL.
db.dir	character, file path for saving the RData, default is db directory under main.dir when setting for main.dir.

Details

This is a pre-processing function for NetBID2 advanced analysis, user only need to input the species name (e.g Homo sapiens, Mus musculus). The function could automatically download information from MSigDB by the functions in msigdbr and save into RData under the db/ directory with specified species name.

Value

Reture TRUE if success and FALSE if not. Will load two variables into R workspace, all_gs2gene and all_gs2gene_info

Examples

```
gs.preload(use_spe='Homo sapiens',update=FALSE)
gs.preload(use_spe='Mus musculus',update=FALSE)
print(all_gs2gene_info)
# contain the information for all gene set category, category info, category size,
## sub category,sub category info, sub category size
print(names(all_gs2gene)) # the first level of the list is the category and sub-category IDs
print(str(all_gs2gene$`CP:KEGG`))

## Not run:
gs.preload(use_spe='Homo sapiens',update=TRUE)

## End(Not run)
```

IQR.filter *IQR (interquartile range) filter for the genes in the expression matrix*

Description

IQR.filter is a function to extract genes by their IQR value.

Usage

```
IQR.filter(exp_mat, use_genes = rownames(exp_mat), thre = 0.5,
  loose_gene = NULL, loose_thre = 0.1)
```

Arguments

exp_mat	matrix, the gene expression matrix, with each row a gene/transcript/probe and each column a sample.
use_genes	a vector of characters, the gene list to report. Default is the rownames of exp_mat.
thre	numeric, the quantile threshold of IQR. Default is 0.5.
loose_gene	a vector of characters, the gene list that only need to pass the loose_thre. This parameter is designed for inputing possible drivers used in SJAracne. Default is NULL.
loose_thre	numeric,the quantile threshold of IQR for the genes in loose_gene. Default is 0.1.

Details

This function aims to extract out most variable genes (defined by the IQR value). This step will be used to perform sample cluster and to prepare the input for SJAracne.

Value

a vector with logical values indicate which genes should be kept.

Examples

```
mat1 <- matrix(rnorm(15000),nrow=1500,ncol=10)
colnames(mat1) <- paste0('Sample',1:ncol(mat1))
rownames(mat1) <- paste0('Gene',1:nrow(mat1))
choose1 <- IQR.filter(mat1,thre=0.5,
                      loose_gene=paste0('Gene',1:100))
```

load.exp.GEO *Load gene expression set from GEO database*

Description

load.exp.GEO is a function to get GEO dataset, save into ExpressionSet class object and save to RData. If the RData exists, user could choose to directly load from it.

Usage

```
load.exp.GEO(out.dir = network.par$out.dir.DATA, GSE = NULL,
  GPL = NULL, getGPL = TRUE, update = FALSE)
```

Arguments

out.dir	character, the file path to save the RData or if the RData has already generated, load from it. Default is network.par$out.dir.DATA if follow the NetBID2 suggested pipeline.
GSE	character, the GSE ID to download.
GPL	character, the GPL ID to download.
getGPL	logical, whether to download the GPL file or not. Default is TRUE.
update	logical, whether to update the RData if the data is already in the out.dir.Default is FALSE

Details

This function aims to provide simple way to get expression set from GEO database and manage the related RData.

Value

ExpressionSet class object if success.

Examples

```
## Not run:
net_eset <- load.exp.GEO(out.dir='test/',
                         GSE='GSE116028',
                         GPL='GPL6480',
                         getGPL=TRUE,
                         update=FALSE)

## End(Not run)
```

load.exp.RNASeq.demo *Load gene expression set from RNASeq results (demo version).*

Description

load.exp.RNASeq.demo is a function to read in RNASeq results and convert to eSet/DESeqDataSet class object.

Usage

```
load.exp.RNASeq.demo(files, type = "salmon", tx2gene = NULL,
  use_phenotype_info = NULL, use_sample_col = NULL,
  use_design_col = NULL, return_type = "eset", merge_level = "gene")
```

Arguments

files	a vector of characters,filenames for the transcript-level abundances, will be passed to tximport. Check for detail.
type	character, he type of software used to generate the abundances, will be passed to tximport. Check for detail.
tx2gene	data.frame or NULL, this parameter will be passed to tximport, check for detail.
use_phenotype_info	data.frame, phenotype information dataframe, must contain the columns use_sample_col and use_design_col.
use_sample_col	character, the column name to indicate which column in use_phenotype_info should be used as the sample name.
use_design_col	character, the column name to indicate which column in use_phenotype_info should be used as the design feature of the samples.
return_type	character, the class of the return object, choose from 'eset','dds','tpm-dds'. 'eset' is the ExpressionSet class object, 'dds' is the DESeqDataSet class object.'tpm-dds' is the original tpm without DESeq2. Default is 'eset'
merge_level	character, choose from 'gene' and 'transcript', if choose 'gene' and original salmon results is mapped to the transcriptome, expression matrix will be merged to gene level. (this only works when using e.g gencode.v29.transcripts.fa from GENCODE as reference).

Details

This function assist to read in RNASeq results from different resources. Due to the complicated condition (e.g reference sequence) in running RNASeq, this function is just a demo version that may not suit for all conditions.

load.exp.RNASeq.demoSalmon *Load gene expression set from Salmon output (demo version).*

Description

load.exp.RNASeq.demoSalmon is a function to read in salmon results and convert to eSet/DESeqDataSet class object.

Usage

```
load.exp.RNASeq.demoSalmon(salmon_dir = "", tx2gene = NULL,
  use_phenotype_info = NULL, use_sample_col = NULL,
  use_design_col = NULL, return_type = "eset", merge_level = "gene")
```

Arguments

salmon_dir	character, the main file directory to save the salmon results.
tx2gene	data.frame or NULL, this parameter will be passed to tximport, check for detail. If NULL, will read the transcript name in one of the salmon output (this only works when using e.g gencode.v29.transcripts.fa from GENCODE as reference).
use_phenotype_info	data.frame, phenotype information dataframe, must contain the columns use_sample_col and use_design_col.
use_sample_col	character, the column name to indicate which column in use_phenotype_info should be used as the sample name.
use_design_col	character, the column name to indicate which column in use_phenotype_info should be used as the design feature of the samples.
return_type	character, the class of the return object, choose from 'eset','dds'. 'eset' is the ExpressionSet class object, 'dds' is the DESeqDataSet class object. Default is 'eset'
merge_level	character, choose from 'gene' and 'transcript', if choose 'gene' and original salmon results is mapped to the transcriptome, expression matrix will be merged to gene level. (this only works when using e.g gencode.v29.transcripts.fa from GENCODE as reference).

Details

This function assist to read in RNASeq results from salmon. Due to the complicated condition (e.g reference sequence) in running salmon, this function is just a demo version that may not suit for all conditions.

merge_eset *Merge two ExpressionSet class object.*

Description

merge_eset is a function to merge two ExpressionSet class objects. If the genes in two objects are same, the expression matrix will be directly merged, otherwise, Z-transformation is performed before merging.

Usage

```
merge_eset(eset1, eset2, group1 = NULL, group2 = NULL,
  group_col_name = "original_group", use_col = NULL,
  remove_batch = FALSE)
```

Arguments

eset1	ExpressionSet class, the first dataset to merge.
eset2	ExpressionSet class, the second dataset to merge.
group1	character, name for the first group.
group2	character, name for the second group.
group_col_name,	character, name for the column indicate the originate of the samples in the phenotype information dataframe in the merged ExpressionSet. Default is 'original_group'.
use_col	a vector of characters, the column names in the phenotype information to be kept in the merged ExpressionSet. If NULL, will use the intersected column names between the two datasets. Default is NULL.
remove_batch	logical, indicate whether or not to remove batch effect between two sample set. Default is FALSE.

Value

an ExressionSet object.

Examples

```
mat1 <- matrix(rnorm(10000),nrow=1000,ncol=10)
colnames(mat1) <- paste0('Sample1_',1:ncol(mat1))
rownames(mat1) <- paste0('Gene',1:nrow(mat1))
eset1 <- generate.eset(exp_mat=mat1)
mat2 <- matrix(rnorm(10000),nrow=1000,ncol=10)
colnames(mat2) <- paste0('Sample2_',1:ncol(mat1))
rownames(mat2) <- paste0('Gene',1:nrow(mat1))
eset2 <- generate.eset(exp_mat=mat2)
new_eset <- merge_eset(eset1,eset2)
```

merge_gs *Merge GeneSets by choosing several categories.*

Description

merge_gs is a simple function to merge the gene set list.

Usage

```
merge_gs(all_gs2gene = all_gs2gene, use_gs = c("H", "CP:BIOCARTA",
  "CP:REACTOME", "CP:KEGG", "C5"))
```

Arguments

all_gs2gene	list, which could be obtained by running gs.preload().
use_gs	a vector of characters, could check all_gs2gene_info for the cateogory description. Default is 'H', 'CP:BIOCARTA', 'CP:REACTOME', 'CP:KEGG'.

Value

a one-level list for geneset to genes.

Examples

```
gs.preload(use_spe='Homo sapiens',update=FALSE)
use_gs2gene <- merge_gs(all_gs2gene=all_gs2gene,
                        use_gs=c('H','CP:BIOCARTA','CP:REACTOME','CP:KEGG','C5'))
```

merge_target_list *Merge target list for two drivers.*

Description

merge_target_list is a function to merge the target list for two drivers. Higher MI statistics for the shared target genes by the two drivers will be kept in the final target list.

Usage

```
merge_target_list(driver1 = NULL, driver2 = NULL, target_list = NULL)
```

Arguments

driver1	character, the name for the first driver to merge.
driver2	character, the name for the second driver to merge.
target_list	a list for the target gene information for the drivers. The names for the list must contain the driver1 and driver2. Each object in the list must be a data.frame and should contain one column ("target") to save the target genes. Strongly suggest to follow the NetBID2 pipeline, and the target_list could be automatically generated by get_net2target_list by running get.SJAracne.network.

Value

a list for the target gene information.

Examples

```
analysis.par <- list()
analysis.par$out.dir.DATA <- system.file('demo1','driver/DATA/',package = "NetBID2")
NetBID.loadRDData(analysis.par=analysis.par,step='ms-tab')
ms_tab <- analysis.par$final_ms_tab
driver1 <- ms_tab[1,'originalID_label']
driver2 <- ms_tab[2,'originalID_label']
m1 <- merge_target_list(driver1=driver1,driver2=driver2,
                        target_list=analysis.par$merge.network$target_list)
```

---

merge_TF_SIG.AC *Merge activity value ExpressionSet for TF (transcription factors) and Sig (signaling factors)*

Description

merge_TF_SIG.AC is a function to merge the activity value ExpressionSet for TF (transcription factors) and Sig (signaling factors).

Usage

```
merge_TF_SIG.AC(TF_AC = NULL, SIG_AC = NULL)
```

Arguments

TF_AC	ExpressionSet object containing the activity value for all TFs.
SIG_AC	ExpressionSet object containing the activity value for all SIGs.

Details

This function aims to merge the driver activity ExpressionSet, by automatically add "_TF"/"_SIG" suffix for drivers.

Value

an ExpressionSet object

Examples

```
analysis.par <- list()
analysis.par$out.dir.DATA <- system.file('demo1','driver/DATA/',package = "NetBID2")
NetBID.loadRDData(analysis.par=analysis.par,step='ms-tab')
ac_mat_TF <- cal.Activity(target_list=analysis.par$tf.network$target_list,
                      cal_mat=exprs(analysis.par$cal.eset),
                      es.method='weightedmean')
ac_mat_SIG <- cal.Activity(target_list=analysis.par$tf.network$target_list,
                      cal_mat=exprs(analysis.par$cal.eset),
                      es.method='weightedmean')
analysis.par$ac.tf.eset  <- generate.eset(exp_mat=ac_mat_TF,
```

```
                                          phenotype_info=pData(analysis.par$cal.eset))
analysis.par$ac.sig.eset <- generate.eset(exp_mat=ac_mat_SIG,
                                          phenotype_info=pData(analysis.par$cal.eset))
analysis.par$merge.ac.eset <- merge_TF_SIG.AC(TF_AC=analysis.par$ac.tf.eset,
                                          SIG_AC=analysis.par$ac.sig.eset)
```

`merge_TF_SIG.network` *Merge target network for TF (transcription factors) and Sig (signaling factors).*

Description

`merge_TF_SIG.network` is a function to merge the target network from TF and Sig.

Usage

```
merge_TF_SIG.network(TF_network = NULL, SIG_network = NULL)
```

Arguments

`TF_network`	TF network obtained by `get.SJAracne.network`.
`SIG_network`	SIG network obtained by `get.SJAracne.network`.

Value

This function will return the same structure object as TF_network/SIG_network, which is a list containing three items, `network_dat`, `target_list` and `igraph_obj`.

Examples

```
if(exists('analysis.par')==TRUE) rm(analysis.par)
network.dir <- sprintf('%s/demo1/network/',system.file(package = "NetBID2")) # use demo
network.project.name <- 'project_2019-02-14' # demo project name
project_main_dir <- 'test/'
project_name <- 'test_driver'
analysis.par  <- NetBID.analysis.dir.create(project_main_dir=project_main_dir,
                                            prject_name=project_name,
                                            network_dir=network.dir,
                                            network_project_name=network.project.name)
analysis.par$tf.network <- get.SJAracne.network(network_file=analysis.par$tf.network.file)
analysis.par$sig.network <- get.SJAracne.network(network_file=analysis.par$sig.network.file)
analysis.par$merge.network <- merge_TF_SIG.network(TF_network=analysis.par$tf.network,
                                                   SIG_network=analysis.par$sig.network)
```

NetBID.analysis.dir.create *Create directory for the driver analysis part. Suggested but not required.*

Description

NetBID.analysis.dir.create will generate a working directory structure for the driver analysis part in NetBID2. This function aims to assist researchers to organize the working directory. It is suggested but not required.

Usage

```
NetBID.analysis.dir.create(project_main_dir = NULL, prject_name = NULL,
  network_dir = NULL, network_project_name = NULL,
  tf.network.file = NULL, sig.network.file = NULL)
```

Arguments

project_main_dir	character, main directory for the project in the driver analysis part.
prject_name	character, project name.
network_dir	character, main directory for the project in the network generation part.
network_project_name	character, the project name of the network for use (SJAR.project_name in SJ.SJAracne.prepare or SJAracne.prepare); This parameter is optional. If previously has not follow the NetBID2 suggested pipeline, could leave this to NULL but set the real path to tf.network.file and sig.network.file if want to follow the driver analysis part of NetBID2 suggested pipeline.
tf.network.file	character, file path of the TF network (XXX/consensus_network_ncol_.txt). Optional, if do not set network_project_name.
sig.network.file	character, file path of the SIG network (XXX/consensus_network_ncol_.txt). Optional, if do not set network_project_name.

Details

This function need to input the main directory for the analysis project with the project name, and need to input the project directory for the network generation with the network_project_name represents the name of the network for use (SJAR.project_name in SJ.SJAracne.prepare) It will generate three sub-directories, QC/ for the QC-related plots, DATA/ for saving the RData, and PLOT/ for saving visulization plots. Such organization is suggested but not required to use all functions in NetBID2. This function will return a variable called analysis.par. This variable is strongly suggested to use in the driver analysis part of NetBID2. It will be used to store all related datasets during calculation.

Value

a list called analysis.par, including main.dir,project.name, out.dir, out.dir.QC, out.dir.DATA, out.dir.PLOT

Examples

```
## Not run:
network.dir <- sprintf('%s/demo1/network/',system.file(package = "NetBID2")) # use demo
network.project.name <- 'project_2019-02-14' #
project_main_dir <- 'demo1/'
project_name <- 'driver_test'
analysis.par  <- NetBID.analysis.dir.create(project_main_dir=project_main_dir,
                                            prject_name=project_name,
                                            network_dir=network.dir,
                                            network_project_name=network.project.name)

## End(Not run)
```

`NetBID.loadRData` *Automatically load RData for NetBID2. Suggested in the NetBID2 pipeline analysis but not required.*

Description

`NetBID.loadRData` is a function strongly suggested to control the pipeline analysis in NetBID2.

Usage

```
NetBID.loadRData(network.par = NULL, analysis.par = NULL,
  step = "exp-load")
```

Arguments

`network.par`	list, store all related datasets during network generation part.
`analysis.par`	list, store all related datasets during driver analysis part.
`step`	character, name for the step.

Details

Users could save two complicate list object, network.par in the network generation part and analysis.par in the driver analysis part, into the data directory (network.par$out.dir.DATA or analysis.par$out.dir.DATA), with the name of the RData marked by `step` name. The two lists could make user to save the whole related dataset in each step `NetBID.saveRData` and easy to get them back by using `NetBID.loadRData`. The RData saved from each step could be used to run the following analysis without repeating the former steps.

Examples

```
analysis.par <- list()
analysis.par$out.dir.DATA <- system.file('demo1','driver/DATA/',package = "NetBID2")
NetBID.loadRData(analysis.par=analysis.par,step='ms-tab')
```

NetBID.network.dir.create *Create directory for the network generation part. Suggested but not required.*

Description

NetBID.network.dir.create will generate a working directory structure for the network generation part in NetBID2. This function aims to assist researchers to organize the working directory. It is suggested but not required.

Usage

```
NetBID.network.dir.create(project_main_dir = NULL, prject_name = NULL)
```

Arguments

project_main_dir	character, main directory for the project.
prject_name	character, project name.

Details

This function need to input the main directory for the project with the project name. It will generate three sub-directories, QC/ for the QC-related plots, DATA/ for saving the RData, and SJAR/ for running SJAracne. Such organization is suggested but not required to use all functions in NetBID2. This function will return a variable called network.par. This variable is strongly suggested to use in the network generation part of NetBID2. It will be used to store all related datasets during calculation.

Value

a list called network.par, including main.dir,project.name, out.dir, out.dir.QC, out.dir.DATA, out.dir.SJAR

Examples

```
## Not run:
NetBID.network.dir.create(project_main_dir='demo1/',project_name='network_test')

## End(Not run)
```

NetBID.saveRData *Automatically save RData for NetBID2. Suggested in the NetBID2 pipeline analysis but not required.*

Description

NetBID.saveRData is a function strongly suggested to control the pipeline analysis in NetBID2.

Usage

```
NetBID.saveRData(network.par = NULL, analysis.par = NULL,
  step = "exp-load")
```

Arguments

network.par	list, store all related datasets during network generation part.
analysis.par	list, store all related datasets during driver analysis part.
step	character, name for the step.

Details

Users could save two complicate list object, network.par in the network generation part and analysis.par in the driver analysis part, into the data directory (network.par$out.dir.DATA or analysis.par$out.dir.DATA), with the name of the RData marked by step name. The two lists could make user to save the whole related dataset in each step NetBID.saveRData and easy to get them back by using NetBID.loadRData. The RData saved from each step could be used to run the following analysis without repeating the former steps.

Examples

```
## Not run:
analysis.par <- list()
analysis.par$out.dir.DATA <- system.file('demo1','driver/DATA/',package = "NetBID2")
NetBID.loadRData(analysis.par=analysis.par,step='ms-tab')
NetBID.saveRData(analysis.par=analysis.par,step='ms-tab_test')

## End(Not run)
```

---

out2excel *Output master table to excel files.*

Description

out2excel is a function to output dataframe into an excel file, mainly for output the master table generated by generate.masterTable.

Usage

```
out2excel(all_ms_tab, out.xlsx, mark_gene = NULL, mark_col = NULL,
  mark_strategy = "color", workbook_name = "ms_tab",
  only_z_sheet = FALSE, z_column = NULL, sig_thre = 1.64)
```

Arguments

all_ms_tab	list or dataframe, if dataframe, it is the master table generated by generate.masterTable. If it is a list, each item in list could be a dataframe containing the master table. The name of the list will be the sheet name in the excel file.
out.xlsx	character, file name for the output excel.
mark_gene	list, list of marker genes, additional info to add in the master table.The name of the list is the name for the mark group.
mark_col	character, the color used to label the marker genes. If NULL, will use get.class.color to get the colors.
mark_strategy	character, choose from 'color','add_column'. 'Color' means the mark_gene will be displayed by its background color; 'add_column' means the mark_gene will be displayed in separate columns with content TRUE/FALSE indicating whether the genes belong to each mark group.
workbook_name	character, workbook name for the output excel. Default is 'ms_tab'.
only_z_sheet	logical, if TRUE will generate a separate sheet only contain Z related columns in DA/DE. Default is FALSE.
z_column	character, the column name that contain the Z-statistics. If NULL, will find columns start with "Z.". Default is NULL.
sig_thre	numeric, threshold for the Z-statistics, values passed the threshold will be marked by the color automatically generated by z2col. Default is 1.64.

Value

logical value indicating whether the file has been sucessfully generated or not.

Examples

```
## Not run:
analysis.par <- list()
analysis.par$out.dir.DATA <- system.file('demo1','driver/DATA/',package = "NetBID2")
NetBID.loadRData(analysis.par=analysis.par,step='ms-tab')
ms_tab <- analysis.par$final_ms_tab ## this is master table generated before
mark_gene <- list(WNT=c('WIF1','TNC','GAD1','DKK2','EMX2'),
                  SHH=c('PDLIM3','EYA1','HHIP','ATOH1','SFRP1'),
                  Group3=c('IMPG2','GABRA5','EGFL11','NRL','MAB21L2','NPR3','MYC'),
                  Group4=c('KCNA1','EOMES','KHDRBS2','RBM24','UNC5D'))
mark_col <- get.class.color(names(mark_gene))
outfile <- 'test_out.xlsx'
out2excel(ms_tab,out.xlsx = out_file,mark_gene,mark_col)

## End(Not run)
```

RNASeqCount.normalize.scale

Simple function to normalize RNASeq Read Count data.

Description

RNASeqCount.normalize.scale is a simple version function to normalize the RNASeq read count data.

Usage

```
RNASeqCount.normalize.scale(mat, total = NULL, pseudoCount = 1)
```

Arguments

mat	matrix, the original input matrix for the read data, each row is a gene/transcript with each column a sample.
total	integer, total read counts, if NULL will use the mean of the column sum. Default is NULL.
pseudoCount	integer, pseudo count to add for all read counts to avoid -Inf in following log transformation. Default is 1.

Details

Strongly suggest to follow the DESeq2 pipeline to process the RNASeq datasets. `load.exp.RNASeq.demo` and `load.exp.RNASeq.demoSalmon` is included in NetBID2 but may not suit for all conditions.

Value

a matrix containing the normalized RNASeq count.

Examples

```
mat1 <- matrix(rnbinom(10000, mu = 10, size = 1),nrow=1000,ncol=10)
colnames(mat1) <- paste0('Sample1',1:ncol(mat1))
rownames(mat1) <- paste0('Gene',1:nrow(mat1))
norm_mat1 <- RNASeqCount.normalize.scale(mat1)
```

SJAracne.prepare *Prepare for running SJaracne*

Description

`SJAracne.prepare` is a function to prepare files for running SJAracne.

Usage

```
SJAracne.prepare(eset, use.samples = rownames(pData(eset)),
  TF_list = NULL, SIG_list = NULL, SJAR.main_dir = "",
  SJAR.project_name = "", IQR.thre = 0.5, IQR.loose_thre = 0.1,
  add_options = "")
```

Arguments

eset	an ExpressionSet class object, which contains the expression matrix.
use.samples	a vector of characters, the sample list used for running SJAracne.
TF_list	a vector of characters, the TF list used for analysis.
SIG_list	a vector of characters, the SIG list used for analysis.
SJAR.main_dir	character, the file path to save the results for SJAracne

SJAR.project_name
: character, the name of the project used to label the output directory.

IQR.thre
: numeric, threshold for IQR filter for all non-driver genes.

IQR.loose_thre
: numeric, threshold for IQR filter for all driver(TF/SIG) genes.

add_options
: additional options used to run sjaracne.

Details

Detailed description to run SJAracne could be found in Github. Check https://github.com/jyyulab/SJARACNe/ for detail.

Examples

```
## Not run:
network.par <- list()
network.par$out.dir.DATA <- system.file('demo1','network/DATA/',package = "NetBID2")
NetBID.loadRData(network.par=network.par,step='exp-QC')
db.preload(use_level='gene',use_spe='human',update=FALSE)
use_gene_type <- 'external_gene_name' ## this should user-defined !!!
use_genes <- rownames(fData(network.par$net.eset))
use_list  <- get.TF_SIG.list(use_genes,use_gene_type=use_gene_type)
#select sample for analysis
phe <- pData(network.par$net.eset)
use.samples <- rownames(phe) ## use all samples, or choose to use some samples
prj.name <- network.par$project.name # can use other names, if need to run different use samples
network.par$out.dir.SJAR <- 'test' ## set the directory
SJAracne.prepare(eset=network.par$net.eset,
                 use.samples=use.samples,
                 TF_list=use_list$tf,
                 SIG_list=use_list$sig,
                 IQR.thre = 0.5,IQR.loose_thre = 0.1,
                 SJAR.project_name=prj.name,
                 SJAR.main_dir=network.par$out.dir.SJAR)

## End(Not run)
```

---

test.targetNet.overlap

Test for the target genes' intersection between two drivers.

Description

test.targetNet.overlap will test whether the target genes of two drivers are significantly intersected.

Usage

```
test.targetNet.overlap(source1_label = NULL, source2_label = NULL,
  target1 = NULL, target2 = NULL, total_possible_target = NULL)
```

Arguments

source1_label	character, the label for the first driver.
source2_label	character, the label for the second driver.
target1	a vector of characters, the list of target genes for the first driver.
target2	a vector of characters, the list of target genes for the second driver.
total_possible_target	numeric or a vector of characters. If input numeric, will be the total number of possible targets. If input a vector of characters, will be the background list of all possible target genes.

Details

This is a function to perform Fisher's Exact Test for the intersection of the target genes from two drivers.

Value

Return the statistics of testing, including the `P.Value`, `Odds_Ratio` and `Intersected_Nuumber`.

Examples

```
source1_label <- 'test1'
target1 <- sample(paste0('G',1:1000),size=80)
source2_label <- 'test2'
target2 <- sample(paste0('G',1:1000),size=120)
test.targetNet.overlap(source1_label=source1_label,source2_label=source2_label,
               target1=target1,target2=target2,
               total_possible_target=paste0('G',1:1000))
## Not run:
```

---

update_eset.feature *Update ExpressionSet feature information, mainly for gene ID conversion*

Description

`update_eset.feature` is a function to update the feature information in the ExpressionSet object.

Usage

```
update_eset.feature(use_eset = NULL, use_feature_info = NULL,
  from_feature = NULL, to_feature = NULL, merge_method = "median",
  distribute_method = "equal")
```

Arguments

use_eset	ExpressionSet class object, the original ExpressionSet to update.
use_feature_info	data.frame, the transfer table for ID conversion.
from_feature	character,the column name in use_feature_info,must be the same type of the rownames for the expression matrix in use_eset.
to_feature	character, the column in use_feature_info, the target ID will be convereted to.
merge_method	character, startegy to merge the expression value, choose from 'median','mean','max','min'. Default is "median".
distribute_method	character, strategy to distribute the expression value, choose from 'mean', 'equal'. Default is "equal".

Details

This function is designed mainly for gene ID conversion. User could input the transfer table for ID conversion, which could be obtained from the original feature table (if use load.exp.GEO and set getGPL==TRUE) or by running the function get_IDtransfer. The relationship betweeen the original ID and the target ID could be as follows: 1) one->one, just change the ID label. 2) multiple->one, the expression value for the target ID will be the merge of the original ID, with user-defined choice of merge_method. 3) one->mulitple, the expression value for the original ID will be distributed to the matched target IDs, with user-defined choice of distribute_method. warning message will be presented in this condition. 4) multiple–>multiple, the function will do distribute step 3) first and merge 2).

Value

an ExressionSet object.

Examples

```
mat1 <- matrix(rnorm(10000),nrow=1000,ncol=10)
colnames(mat1) <- paste0('Sample',1:ncol(mat1))
rownames(mat1) <- paste0('Gene',1:nrow(mat1))
eset <- generate.eset(exp_mat=mat1)
test_transfer_table <- data.frame(
                      'Gene'=c('Gene1','Gene1','Gene2','Gene3','Gene4'),
                      'Transcript'=c('T11','T12','T2','T3','T3'))
new_eset <- update_eset.feature(use_eset=eset,
                                use_feature_info=test_transfer_table,
                                from_feature='Gene',
                                to_feature='Transcript',
                                merge_method='median',
                                distribute_method='equal'
                                )
print(exprs(eset)[test_transfer_table$Gene,])
print(exprs(new_eset))
```

update_eset.phenotype *Update ExpressionSet phenotype information, mainly for changing sample name and extract useful phenotype information for sample clustering.*

Description

update_eset.phenotype is a function to update the phenotype information in the ExpressionSet object.

Usage

```
update_eset.phenotype(use_eset = NULL, use_phenotype_info = NULL,
  use_sample_col = NULL, use_col = "auto")
```

Arguments

use_eset	ExpressionSet class object, the original ExpressionSet to update.
use_phenotype_info	data.frame, the phenotype information dataframe.
use_sample_col	character,the column name in use_phenotype_info, which contains the sample name.
use_col	character, the columns in use_phenotype_info to be kept. If set to 'auto', wil extract columns with unique sample feature ranges from 2 to sample size-1. If set to 'GEO-auto', will extract columns: 'geo_accession','title','source_name_ch1',and columns end with ':ch1'. Default is "auto".

Details

This function is designed mainly for extracting useful phenotype information for samples. Especially for phenotype information directly get from GEO database.

Value

an ExressionSet object.

Examples

```
## Not run:
net_eset <- load.exp.GEO(out.dir='test/',
                         GSE='GSE116028',
                         GPL='GPL6480',
                         getGPL=TRUE,
                         update=FALSE)
net_eset <- update_eset.phenotype(use_eset=net_eset,
                                  use_phenotype_info=pData(net_eset),
                                  use_sample_col='geo_accession',
                                  use_col='GEO-auto')

## End(Not run)
```

update_SJAracne.network *Update the network information in the structured network list dataset*

Description

update_SJAracne.network is a function to update the network information by input threshold for statistics or input gene list for use.

Usage

```
update_SJAracne.network(network_list = NULL,
  all_possible_drivers = NULL, all_possible_targets = NULL,
  force_all_drivers = TRUE, force_all_targets = TRUE, min_MI = 0,
  max_p.value = 1, min_spearman_value = 0, min_pearson_value = 0,
  spearman_sign_use = c(1, -1), pearson_sign_use = c(1, -1),
  directed = TRUE, weighted = TRUE)
```

Arguments

network_list	list,the network list dataset generated by get.SJAracne.network, contains network_dat, target_list and igraph_obj.
all_possible_drivers	a vector of characters,all possible driver list used to filter the network file. If NULL, will set to the possible drivers from network_list. Default is NULL.
all_possible_targets	a vector of characters,all possible target list used to filter the network file. If NULL, will set to the possible targets network_list. Default is NULL.
force_all_drivers	logical, whether or not to include all genes in the all_possible_drivers in the final network. For network_dat and target_list, if force_all_drivers is set to TRUE, genes in all_possible_drivers will not be filtered by the following statistical thresholds. For igraph_obj, if force_all_drivers is set to TRUE, all genes in all_possible_drivers, even not exist in the original network file will be include in the vertice of the igraph object. Default is TRUE.
force_all_targets	logical, whether or not to include all genes in the all_possible_targets in the final network. For network_dat and target_list, if all_possible_targets is set to TRUE, genes in all_possible_targets will not be filtered by the following statistical thresholds. For igraph_obj, if force_all_targets is set to TRUE, all genes in all_possible_targets, even not exist in the original network file will be include in the vertice of the igraph object. Default is TRUE.
min_MI	numeric, minimum threshold for MI. Default is 0.
max_p.value	numeric, maximum threshold for p.value. Default is 1.
min_spearman_value	numeric, minimum threshold for spearman absolute value. Default is 0.
min_pearson_value	numeric, minimum threshold for pearson absolute value. Default is 0.

spearman_sign_use
: a vector of numeric value, 1 indicates positve values in spearman will be used. -1 indicates negative values will be used. If only want to include positive values, set spearman_sign_use to 1. Default is c(1,-1).

pearson_sign_use
: a vector of numeric value, 1 indicates positve values in pearson will be used. -1 indicates negative values will be used. If only want to include positive values, set pearson_sign_use to 1. Default is c(1,-1).

directed
: logical, whether the network in igraph is directed or not. Default is TRUE.

weighted
: logical, whether to add the edge weight in the igraph object. Default is TRUE.

Details

This function aims to update the network list dataset generated by get.SJAracne.network and return the list dataset passed the filtration with the same data structure.

Value

This function will return a list containing three items, network_dat, target_list and igraph_obj.

Examples

```
if(exists('analysis.par')==TRUE) rm(analysis.par)
network.dir <- sprintf('%s/demo1/network/',system.file(package = "NetBID2")) # use demo
network.project.name <- 'project_2019-02-14' # demo project name
project_main_dir <- 'test/'
project_name <- 'test_driver'
analysis.par  <- NetBID.analysis.dir.create(project_main_dir=project_main_dir,
                                            prject_name=project_name,
                                            network_dir=network.dir,
                                            network_project_name=network.project.name)
analysis.par$tf.network <- get.SJAracne.network(network_file=analysis.par$tf.network.file)
analysis.par$sig.network <- get.SJAracne.network(network_file=analysis.par$sig.network.file)
all_possible_drivers <- c(names(analysis.par$tf.network$target_list)[1:1000],
                          'addition_driver_1','addition_driver_2')
tf.network.update <- update_SJAracne.network(network_list=analysis.par$tf.network,
                                             all_possible_drivers=all_possible_drivers,
                                             force_all_drivers=TRUE,
                                             force_all_targets=FALSE,
                                             pearson_sign_use=1)
print(intersect(c('addition_driver_1','addition_driver_2'),
      names(V(tf.network.update$igraph_obj)))) ## check
## Not run:
```

---

z2col *Get the color for the input Z statistics.*

Description

z2col is a function to transfer the input Z statistics to a color bar.

Usage

```
z2col(x, n_len = 60, sig_thre = 0.01, col_min_thre = 0.01,
  col_max_thre = 3, blue_col = brewer.pal(9, "Set1")[2],
  red_col = brewer.pal(9, "Set1")[1])
```

Arguments

x	a vector of numeric values. The input Z statistics.
n_len	integer, number of unique colors. Default is 60.
sig_thre	numeric, the threshold for significance (absolute Z statistics), values do not pass the threshold will be colored in 'white'.
col_min_thre	numeric, the threshold for the lowest values used to generate the color bar. Default is 0.01.
col_max_thre	numeric, the threshold for the maximum values used to generate the color bar. Default is 3.
blue_col	a vector of characters, the blue colors used for the negative values in Z statistics. Default is brewer.pal(9,'Set1')[2].
red_col	a vector of characters, the red lors used for the negative values in Z statistics. Default is brewer.pal(9,'Set1')[1].

Value

a vector of color characters

Examples

```
t1 <- sort(rnorm(mean=0,sd=2,n=100))
image(as.matrix(t1),col=z2col(t1))
```

Index