Package 'NetBID2'

January 11, 2021

Type Package

Title Network-based Bayesian Inference of Drivers, version 2

Version 0.1.2

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

Description An integrative systems biology algorithm to infer drivers of phenotype based on data-driven context-specific network and Bayesian inference.

License GPL (>= 2)

Encoding UTF-8

LazyData true

Depends R (>= 3.6.0),
Matrix (>= 1.2-17),
SummarizedExperiment (>= 1.14.0),
lme4 (>= 1.1-21),
Biobase (>= 2.38.0),
GEOquery (>= 2.46.15),
limma (>= 3.34.9),
impute (>= 1.52.0),
tximport (>= 1.6.0),
DESeq2 (>= 1.20.0),
ConsensusClusterPlus (>= 1.38.0),
aricode (>= 0.1.1),
igraph (>= 1.1.2),
RColorBrewer (>= 1.1-2),
plot3D (>= 1.1.1),
plotrix (>= 3.7-3),
umap (>= 0.2.0.0),
ComplexHeatmap (>= 1.17.1),
ordinal (>= 2019.4-25),
MCMCglmm (>= 2.26),
arm (>= 1.10-1),
reshape (>= 0.8.7),
biomaRt (>= 2.34.2),
GSVA (>= 1.22.4),
msigdbr (>= 6.2.1),
openxlsx (>= 4.1.0),
rmarkdown (>= 1.11),
kableExtra (>= 1.1.0),
rhdf5 (>= 2.28.0),

plotly (>= 4.9.0)

Imports

Suggests

RoxygenNote 7.1.1

R topics documented:

`bid` *Calculate Differential Expression (DE) or Differential Activity (DA) by Using Bayesian Inference*

Description

`bid` calculates the differential expression (DE) / differential activity (DA) by using Bayesian Inference method. Users can choose different regression models and pooling strategies.

`bid` calculates the differential expression (DE) / differential activity (DA) by using Bayesian Inference method. Users can choose different regression models and pooling strategies.

Usage

```
bid(
  mat = NULL,
  use_obs_class = NULL,
  class_order = NULL,
  class_ordered = TRUE,
  method = "Bayesian",
```

```
  family = gaussian,
  pooling = "full",
  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
)

bid(
  mat = NULL,
  use_obs_class = NULL,
  class_order = NULL,
  class_ordered = TRUE,
  method = "Bayesian",
  family = gaussian,
  pooling = "full",
  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, the expression/activity matrix of IDs (gene/transcript/probe) from one gene. Rows are IDs, columns are samples. It is strongly suggested to contain rownames of IDs and column names of samples. Example, geneA has two probes A1 and A2 across all 6 samples (Case-rep1, Case-rep2, Case-rep3, Control-rep1, Control-rep2 and Control-rep3). The `mat` of geneA is a 2*6 numeric matrix. Likewise, if geneA has only one probe, the `mat` is a one-row matrix.
`use_obs_class`	a vector of characters, the category of sample. If the vector names are not avail-

able, the order of samples in use_obs_class must be the same as in mat. Users can call get_obs_label to create this vector.

class_order — a vector of characters, the order of the sample's category. The first class in this vector will be considered as the control group by default. If NULL, the order will be assigned using alphabetical order. Default is NULL.

class_ordered — logical, if TRUE, the class_order will be ordered. And the order must be consistent with the phenotypic trend, such as "low", "medium", "high". Default is TRUE.

method — character, users can choose between "MLE" and "Bayesian". "MLE", the maximum likelihood estimation, will call generalized linear model(glm/glmer) to perform data regression. "Bayesian", will call Bayesian generalized linear model (bayesglm) or multivariate generalized linear mixed model (MCMCglmm) to perform data regression. Default is "Bayesian".

family — character or family function or the result of a call to a family function. This parameter is used to define the model's error distribution. See ?family for details. Currently, options are gaussian, poisson, binomial(for two-group sample classes)/category(for multi-group sample classes)/ordinal(for multi-group sample classes with class_ordered=TRUE). If set with gaussian or poission, the response variable in the regression model will be the expression level, and the independent variable will be the sample's phenotype. If set with binomial, the response variable in the regression model will be the sample phenotype, and the independent variable will be the expression level. For binomial, category and ordinal input, the family will be automatically reset, based on the sample's class level and the setting of class_ordered. Default is gaussian.

pooling — character, users can choose from "full","no" and "partial". "full", use probes as independent observations. "no", use probes as independent variables in the regression model. "partial", use probes as random effect in the regression model. Default is "full".

prior.V.scale — numeric, the V in the parameter "prior" used in MCMCglmm. It is meaningful to set when one choose "Bayesian" as method and "partial" as pooling. Default is 0.02.

prior.R.nu — numeric, the R-structure in the parameter "prior" used in MCMCglmm. It is meaningful to set when one choose "Bayesian" as method and "partial" as pooling. Default is 1.

prior.G.nu — numeric, the G-structure in the parameter "prior" used in MCMCglmm. It is meaningful to set when one choose "Bayesian" as method and "partial" as pooling. Default is 2.

nitt — numeric, the parameter "nitt" used in MCMCglmm. It is meaningful to set when one choose "Bayesian" as method and "partial" as pooling. Default is 13000.

burnin — numeric, the parameter "burnin" used in MCMCglmm. It is meaningful to set when one choose "Bayesian" as method and "partial" as pooling. Default is 3000.

thin — numeric, the parameter "thin" used in MCMCglmm. It is meaningful to set when one choose "Bayesian" as method and "partial" as pooling. Default is 10.

std — logical, if TRUE, the expression matrix will be normalized by column. Default is TRUE.

logTransformed — logical, if TRUE, log transformation has been performed. Default is TRUE.

log.base — numeric, the base of log transformation when do.logtransform is set to TRUE. Default is 2.

average.method	character, the method applied to calculate FC (fold change). Users can choose between "geometric" and "arithmetic". Default is "geometric".
pseudoCount	integer, the integer added to avoid "-Inf" showing up during log transformation in the FC (fold change) calculation.
return_model	logical, if TRUE, the regression model will be returned; Otherwise, just return basic statistics from the model. Default is FALSE.
use_seed	integer, the random seed. Default is 999.
verbose	logical, if TRUE, print out additional information during calculation. Default is FALSE.

Details

It is a core function inside getDE.BID.2G. This function allows users to have access to more options when calculating the statistics using Bayesian Inference method. In some cases, the input expression matrix could be at probe/transcript level, but DE/DA calculated at gene level is expected. By setting pooling strategy, users can successfully solve the special cases. The P-value is estimated by the posterior distribution of the coefficient.

It is a core function inside getDE.BID.2G. This function allows users to have access to more options when calculating the statistics using Bayesian Inference method. In some cases, the input expression matrix could be at probe/transcript level, but DE/DA calculated at gene level is expected. By setting pooling strategy, users can successfully solve the special cases. The P-value is estimated by the posterior distribution of the coefficient.

Value

Return a one-row data frame with calculated statistics for one gene/gene set if return_model is FALSE. Otherwise, the regression model will be returned.

Return a one-row data frame with calculated statistics for one gene/gene set if return_model is FALSE. Otherwise, the regression model will be returned.

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:

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 Each Driver*

Description

`cal.Activity` calculates the activity value for each driver. This function requires two inputs, the driver-to-target list object `target_list` and the expression matrix.

`cal.Activity` calculates the activity value for each driver. This function requires two inputs, the driver-to-target list object `target_list` and the expression matrix.

Usage

```
cal.Activity(
  target_list = NULL,
  igraph_obj = NULL,
  cal_mat = NULL,
  es.method = "weightedmean",
  std = TRUE,
  memory_constrain = FALSE
)

cal.Activity(
  target_list = NULL,
  igraph_obj = NULL,
  cal_mat = NULL,
  es.method = "weightedmean",
  std = TRUE,
  memory_constrain = FALSE
)
```

Arguments

`target_list`	list, the driver-to-target list object. Either igraph_obj or target_list is necessary for this function. The names of the list elements are drivers. Each element is a data frame, usually contains at least three columns. "target", target gene names; "MI", mutual information; "spearman", spearman correlation coefficient. "MI" and "spearman" is necessary if es.method="weightedmean". Users can call `get.SJAracne.network` to get this list from the network file generated by SJAracne (the second element of the return list) or prepare the list object by hand but should match the data format described above.
`igraph_obj`	igraph object, optional. Either igraph_obj or target_list is necessary for this function. Users can call `get.SJAracne.network` to get this object from the network file generated by SJAracne (the third element of the return list), or prepare the igraph network object by hand (Directed network and the edge attributes should include "weight" and "sign" if es.method="weightedmean").
`cal_mat`	numeric matrix, the expression matrix of genes/transcripts.
`es.method`	character, method applied to calculate the activity value. User can choose from "mean", "weightedmean", "maxmean" and "absmean". Default is "weightedmean".

`std` logical, if TRUE, the expression matrix will be normalized by column. Default is TRUE.

`memory_constrain` logical, if TRUE, the calculation strategy will not use Matrix Cross Products, which is memory consuming. Default is FALSE.

Value

Return a matrix of activity values. Rows are drivers, columns are samples.

Return a matrix of activity values. Rows are drivers, columns are samples.

Examples

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

---

cal.Activity.GS *Calculate Activity Value for Gene Sets*

Description

`cal.Activity.GS` calculates activity value for each gene set, and return a numeric matrix with rows of gene sets and columns of samples.

`cal.Activity.GS` calculates activity value for each gene set, and return a numeric matrix with rows of gene sets and columns of samples.

Usage

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

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	list, contains elements of gene sets. Element name is gene set name, each element contains a vector of genes belong to that gene set. Default is using all_gs2gene[c('H','CP:BIOCARTA','CP:REACTOME','CP:KEGG')], which is loaded from gs.preload.
cal_mat	numeric matrix, gene/transcript expression matrix. If want to input activity matrix, need to use 'processDriverProfile()' to pre-process the dataset. Detailed could see demo.
es.method	character, method to calculate the activity value. Users can choose from "mean", "absmean", "maxmean", "gsva", "ssgsea", "zscore" and "plage". The details for using the last four options, users can check gsva. Default is "mean".
std	logical, if TRUE, the expression matrix will be normalized by column. Default is TRUE.

Value

Return an activity matrix with rows of gene sets and columns of samples.

Return an activity matrix with rows of gene sets and columns of samples.

Examples

```
analysis.par <- list()
analysis.par$out.dir.DATA <- system.file('demo1','driver/DATA/',package = "NetBID2")
NetBID.loadRData(analysis.par=analysis.par,step='ms-tab')
gs.preload(use_spe='Homo sapiens',update=FALSE)
use_gs2gene <- merge_gs(all_gs2gene=all_gs2gene,
                        use_gs=c('H','CP:BIOCARTA','CP:REACTOME','CP:KEGG','C5'))
exp_mat_gene <- Biobase::exprs(analysis.par$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)
## if want to input activity-matrix
ac_mat <- cal.Activity(target_list=analysis.par$merge.network$target_list,
          cal_mat=Biobase::exprs(analysis.par$cal.eset),
          es.method='weightedmean')
# pre-process the activity matrix by selecting the one
# with larger target size for duplicate drivers
Driver_name <- rownames(ac_mat)
ms_tab <- analysis.par$final_ms_tab
driver_size <- ms_tab[Driver_name,]$Size
use_driver <- processDriverProfile(Driver_profile=driver_size,
                                Driver_name=Driver_name,
                                choose_strategy='max',
                                return_type='driver_name')
use_driver_gene_name <-
              processDriverProfile(Driver_profile=driver_size,
                                Driver_name=Driver_name,
```

```
                              choose_strategy='max',
                              return_type='gene_name')
ac_mat_gene <- ac_mat[use_driver,]
rownames(ac_mat_gene) <- use_driver_gene_name
driver_ac_gs <- cal.Activity.GS(use_gs2gene = use_gs2gene,
                        cal_mat = ac_mat_gene)
analysis.par <- list()
analysis.par$out.dir.DATA <- system.file('demo1','driver/DATA/',package = "NetBID2")
NetBID.loadRData(analysis.par=analysis.par,step='ms-tab')
gs.preload(use_spe='Homo sapiens',update=FALSE)
use_gs2gene <- merge_gs(all_gs2gene=all_gs2gene,
                        use_gs=c('H','CP:BIOCARTA','CP:REACTOME','CP:KEGG','C5'))
exp_mat_gene <- Biobase::exprs(analysis.par$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)
## if want to input activity-matrix
ac_mat <- cal.Activity(target_list=analysis.par$merge.network$target_list,
          cal_mat=Biobase::exprs(analysis.par$cal.eset),
          es.method='weightedmean')
# pre-process the activity matrix by selecting the one
# with larger target size for duplicate drivers
Driver_name <- rownames(ac_mat)
driver_size <- ms_tab[Driver_name,]$Size
use_driver <- processDriverProfile(Driver_profile=driver_size,
                              Driver_name=Driver_name,
                              choose_strategy='max',
                              return_type='driver_name')
use_driver_gene_name <-
              processDriverProfile(Driver_profile=driver_size,
                              Driver_name=Driver_name,
                              choose_strategy='max',
                              return_type='gene_name')
ac_mat_gene <- ac_mat[use_driver,]
rownames(ac_mat_gene) <- use_driver_gene_name
driver_ac_gs <- cal.Activity.GS(use_gs2gene = use_gs2gene,
                        cal_mat = ac_mat_gene)
```

`combineDE` *Combine Multiple Comparison Results from Differential Expression (DE) or Differential Activity (DA) Analysis*

Description

`combineDE` combines multiple comparisons of DE or DA analysis. Can combine DE with DE, DA with DA and also DE with DA if proper transfer table prepared.

`combineDE` combines multiple comparisons of DE or DA analysis. Can combine DE with DE, DA with DA and also DE with DA if proper transfer table prepared.

Usage

```
combineDE(
  DE_list,
  DE_name = NULL,
```

```
  transfer_tab = NULL,
  main_id = NULL,
  method = "Stouffer",
  twosided = TRUE,
  signed = TRUE
)

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

Arguments

DE_list	list, each element in the list is one DE/DA comparison need to be combined.
DE_name	a vector of characters, the DE/DA comparison names. If not NULL, it must match the names of DE_list in correct order. If NULL, names of the DE_list will be used. Default is NULL.
transfer_tab	data.frame, the ID conversion table. Users can call `get_IDtransfer` to get this table. The purpose is to correctly mapping ID for `DE_list`. The column names must match `DE_name`. If NULL, ID column of each DE comparison will be considered as the same type. Default is NULL.
main_id	character, a name of the element in `DE_list`. The ID column of that comparison will be used as the ID of the final combination. If NULL, the first element name from `DE_list` will be used. Default is NULL.
method	character, users can choose between "Stouffer" and "Fisher". Default is "Stouffer".
twosided	logical, if TRUE, a two-tailed test will be performed. If FALSE, a one-tailed test will be performed, and P value falls within the range of 0 to 0.5. Default is TRUE.
signed	logical, if TRUE, give a sign to the P value, which indicating the direction of testing. Default is TRUE.

Details

For example, there are 4 subgroups in the phenotype, G1, G2, G3 and G4. One DE analysis was performed on G1 vs. G2, and another DE was performed on G1 vs. G3. If user is interested in the DE analysis between G1 vs. (G2 and G3), he can call this function to combine the two comparison results above toghether. The combined P values will be taken care by `combinePvalVector`.

For example, there are 4 subgroups in the phenotype, G1, G2, G3 and G4. One DE analysis was performed on G1 vs. G2, and another DE was performed on G1 vs. G3. If user is interested in the DE analysis between G1 vs. (G2 and G3), he can call this function to combine the two comparison results above toghether. The combined P values will be taken care by `combinePvalVector`.

Value

Return a list contains the combined DE/DA analysis. Each single comparison result before combination is wrapped inside (may have with some IDs filtered out, due to the combination). A data frame named "combine" inside the list is the combined analysis. Rows are genes/drivers, columns are combined statistics (e.g. "logFC", "AveExpr", "t", "P.Value" etc.).

Return a list contains the combined DE/DA analysis. Each single comparison result before combination is wrapped inside (may have with some IDs filtered out, due to the combination). A data frame named "combine" inside the list is the combined analysis. Rows are genes/drivers, columns are combined statistics (e.g. "logFC", "AveExpr", "t", "P.Value" etc.).

Examples

```
analysis.par <- list()
analysis.par$out.dir.DATA <- system.file('demo1','driver/DATA/',package = "NetBID2")
NetBID.loadRDData(analysis.par=analysis.par,step='ms-tab')
phe_info <- Biobase::pData(analysis.par$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,main_id='DA')

## 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)
analysis.par <- list()
analysis.par$out.dir.DATA <- system.file('demo1','driver/DATA/',package = "NetBID2")
NetBID.loadRDData(analysis.par=analysis.par,step='ms-tab')
phe_info <- Biobase::pData(analysis.par$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,main_id='DA')

## 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 P Values Using Fisher's Method or Stouffer's Method*

Description

`combinePvalVector` is a function to combine multiple comparison's P values using Fisher's method or Stouffer's method.

`combinePvalVector` is a function to combine multiple comparison's P values using Fisher's method or Stouffer's method.

Usage

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

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

Arguments

pvals	a vector of numerics, the P values from multiple comparison need to be combined.
method	character, users can choose between "Stouffer" and "Fisher". Default is "Stouffer".
signed	logical, if TRUE, will give a sign to the P value to indicate the direction of testing. Default is TRUE.
twosided	logical, if TRUE, P value is calculated in a one-tailed test. If FALSE, P value is calculated in a two-tailed test, and it falls within the range 0 to 0.5. Default is TRUE.

Value

Return a vector contains the "Z-statistics" and "P.Value".

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

---

db.preload *Preload database files into R workspace for NetBID2*

Description

db.preload is a pre-processing function for NetBID2. It preloads needed data into R workspace, and saves it locally under db/ directory with specified species name and analysis level.

db.preload is a pre-processing function for NetBID2. It preloads needed data into R workspace, and saves it locally under db/ directory with specified species name and analysis 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)
)

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, users can choose "transcript" or "gene". Default is "gene".
use_spe	character, the name of an interested species (e.g. "human", "mouse", "rat"). Default is "human".
update	logical, if TRUE, previous loaded RData will be updated. Default is FALSE.
TF_list	a character vector, the list of TF (Transcription Factor) names. If NULL, the pre-defined list in the package will be used. Default is NULL.
SIG_list	a character vector, the list of SIG (Signaling Factor) names. If NULL, the pre-defined list in the package will be use. Default is NULL.
input_attr_type	character, the type of the TF_list and SIG_list. Details please check biomaRt, https://bioconductor.org/packages/release/bioc/vignettes/biomaRt/inst/doc/biomaRt.html. If TF_list and SIG_list are not specified, the list in the NetBID2 package will be used. This only support "external_gene_name" and "ensembl_gene_id". Default is "external_gene_name".
main.dir	character, the main directory for NetBID2. If NULL, will be `system.file(package = "NetBID2")`. Default is NULL.
db.dir	character, a path for saving the RData. Default is db directory under the `main.dir`, if `main.dir` is provided.

Details

Users need to set the species name (e.g. human, mouse) and analysis level (transcript or gene level). TF list and SIG list are optional, if not specified, list from package data will be used as default.

Users need to set the species name (e.g. human, mouse) and analysis level (transcript or gene level). TF list and SIG list are optional, if not specified, list from package data will be used as default.

Value

Return TRUE if loading is successful, otherwise return FALSE. Two variables will be loaded into R workspace, `tf_sigs` and `db_info`.

Return TRUE if loading is successful, otherwise return FALSE. Two variables will be loaded 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)
```

```
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` *Visualize Sample Clustering Result in 2D Plot*

Description

`draw.2D` creats a 2D plot to visualize the sample clustering result.

`draw.2D` creats a 2D plot to visualize the sample clustering result.

Usage

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

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

Arguments

`X`	a vector of numerics, the x coordinates of points in the plot. If user would like to create a PCA biplot, this parameter should be the first component.
`Y`	a vector of numerics, the y coordinates of points in the plot. If user would like to create a PCA biplot, this parameter should be the second component.

class_label	a vector of characters, labels or categories of samples. The vector name should be sample names.
xlab	character, the label for x-axis. Default is "PC1".
ylab	character, the label for y-axis. Default is "PC2".
legend_cex	numeric, giving the amount by which the text of legend should be magnified relative to the default. Default is 0.8.
main	character, an overall title for the plot. Default is "".
point_cex	numeric, giving the amount by which the size of the data points should be magnified relative to the default. Default is 1.
use_color	a vector of color codes, colors to be assigned to each member of display label. Default is brewer.pal(9, 'Set1').
pre_define	a vector of characters, pre-defined color codes for a certain input (e.g. c("blue", "red") with names c("A", "B")). Default is NULL.

Value

Return a logical value. If TRUE, the plot has been created successfully.

Return a logical value. If TRUE, the plot has been created successfully.

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 <- stats::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)
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 <- stats::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 *Visualize Sample Clustering Result in 2D Plot with Ellipse*

Description

draw.2D.ellipse creates a 2D plot with an ellipse drawn around each cluster to visualize the sample clustering result.

draw.2D.ellipse creates a 2D plot with an ellipse drawn around each cluster to visualize the sample clustering result.

Usage

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

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

Arguments

X	a vector of numerics, the x coordinates of points in the plot. If user would like to creat a PCA biplot, this parameter should be the first component.
Y	a vector of numerics, the y coordinates of points in the plot. If user would like to creat a PCA biplot, this parameter should be the second component.
class_label	a vector of characters, labels or categories of samples. The vector name should be sample names.
xlab	character, the label for x-axis. Default is "PC1".
ylab	character, the label for y-axis. Default is "PC2".
legend_cex	numeric, giving the amount by which the text of legend should be magnified relative to the default. Default is 0.8.
main	character, an overall title for the plot. Default is "".
point_cex	numeric, giving the amount by which the size of the data points should be magnified relative to the default. Default is 1.
use_color	a vector of color codes, colors to be assigned to each member of display label. Default is brewer.pal(9, 'Set1').
pre_define	a vector of characters, pre-defined color codes for a certain input (e.g. c("blue", "red") with names c("A", "B")). Default is NULL.

Value

Return a logical value. If TRUE, the plot has been created successfully.

Return a logical value. If TRUE, the plot has been created successfully.

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 <- stats::prcomp(t(mat1))$x
pred_label <- stats::kmeans(pc,centers=4)$cluster ## this can use other cluster results
draw.2D.ellipse(X=pc[,1],Y=pc[,2],class_label=pred_label)
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 <- stats::prcomp(t(mat1))$x
pred_label <- stats::kmeans(pc,centers=4)$cluster ## this can use other cluster results
draw.2D.ellipse(X=pc[,1],Y=pc[,2],class_label=pred_label)
```

---

draw.2D.interactive *Visualize Sample Clustering Result in 2D Plot with interactive mode*

Description

draw.2D.interactive creats a 2D plot to visualize the sample clustering result with interactive mode realized by plotly.

draw.2D.interactive creats a 2D plot to visualize the sample clustering result with interactive mode realized by plotly.

Usage

```
draw.2D.interactive(
  X,
  Y,
  sample_label = NULL,
  color_label = NULL,
  shape_label = NULL,
  xlab = "PC1",
  ylab = "PC2",
  main = "",
  point_cex = 1,
  use_color = NULL,
  pre_define = NULL
)

draw.2D.interactive(
  X,
  Y,
  sample_label = NULL,
  color_label = NULL,
  shape_label = NULL,
  xlab = "PC1",
  ylab = "PC2",
  main = "",
  point_cex = 1,
  use_color = NULL,
```

```
  pre_define = NULL
)
```

Arguments

X	a vector of numerics, the x coordinates of points in the plot. If user would like to create a PCA biplot, this parameter should be the first component.
Y	a vector of numerics, the y coordinates of points in the plot. If user would like to create a PCA biplot, this parameter should be the second component.
sample_label	a vector of characters, name of samples to be displayed on the figure.
color_label	a vector of characters, labels used to define the point color.
shape_label	a vector of characters, labels used to define the point shape.
xlab	character, the label for x-axis. Default is "PC1".
ylab	character, the label for y-axis. Default is "PC2".
main	character, an overall title for the plot. Default is "".
point_cex	numeric, giving the amount by which the size of the data points should be magnified relative to the default. Default is 1.
use_color	a vector of color codes, colors to be assigned to each member of display label. Default is brewer.pal(9, 'Set1').
pre_define	a vector of characters, pre-defined color codes for a certain input (e.g. c("blue", "red") with names c("A", "B")). Default is NULL.

Value

Return the plotly class object for interactive visualization.

Return the plotly class object for interactive visualization.

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 <- stats::prcomp(t(mat1))$x
pred_label <- kmeans(pc,centers=4)$cluster ## this can use other cluster results
draw.2D.interactive(X=pc[,1],Y=pc[,2],
                    sample_label=rownames(pc),
                    color_label=pred_label,
                    pre_define = c('1'='blue','2'='red','3'='yellow','4'='green'))
draw.2D.interactive(X=pc[,1],Y=pc[,2],
                    sample_label=rownames(pc),
                    shape_label=pred_label)
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 <- stats::prcomp(t(mat1))$x
pred_label <- kmeans(pc,centers=4)$cluster ## this can use other cluster results
draw.2D.interactive(X=pc[,1],Y=pc[,2],
                    sample_label=rownames(pc),
                    color_label=pred_label,
                    pre_define = c('1'='blue','2'='red','3'='yellow','4'='green'))
draw.2D.interactive(X=pc[,1],Y=pc[,2],
                    sample_label=rownames(pc),
                    shape_label=pred_label)
```

draw.2D.text *Visualize Sample Clustering Result in 2D Plot with Sample Names*

Description

draw.2D.text creates a 2D plot with sample names labeled, to visualize the sample clustering result.

draw.2D.text creates a 2D plot with sample names labeled, to visualize the sample clustering result.

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,
  use_color = NULL,
  pre_define = NULL
)

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,
  use_color = NULL,
  pre_define = NULL
)
```

Arguments

X	a vector of numerics, the x coordinates of points in the plot. If user would like to create a PCA biplot, this parameter should be the first component.
Y	a vector of numerics, the y coordinates of points in the plot. If user would like to create a PCA biplot, this parameter should be the second component.
class_label	a vector of characters, labels or categories of samples. The vector name should be sample names.

class_text	a vector of characters, the user-defined sample names to label each data points in the plot. If NULL, will use the names of class_label. Default is NULL.
xlab	character, the label for x-axis. Default is "PC1".
ylab	character, the label for y-axis. Default is "PC2".
legend_cex	numeric, giving the amount by which the text of legend should be magnified relative to the default. Default is 0.8.
main	character, an overall title for the plot. Default is "".
point_cex	numeric, giving the amount by which the size of the data points should be magnified relative to the default. Default is 1.
text_cex	numeric, giving the amount by which the text of class_text should be magnified relative to the default. Default is NULL.
use_color	a vector of color codes, colors to be assigned to each member of display label. Default is brewer.pal(9, 'Set1').
pre_define	a vector of characters, pre-defined color codes for a certain input (e.g. c("blue", "red") with names c("A", "B")). Default is NULL.

Value

Return a logical value. If TRUE, the plot has been created successfully.

Return a logical value. If TRUE, the plot has been created successfully.

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 <- stats::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)
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 <- stats::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 *Visualize Sample Clustering Result in 3D Plot*

Description

draw.3D creates a 3D plot to visualize the sample clustering result.

draw.3D creates a 3D plot to visualize the sample clustering result.

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,
  use_color = NULL,
  pre_define = NULL,
  ...
)

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,
  use_color = NULL,
  pre_define = NULL,
  ...
)
```

Arguments

X	a vector of numerics, the x coordinates of points in the plot. If user would like to create a PCA biplot, this parameter should be the first component.
Y	a vector of numerics, the y coordinates of points in the plot. If user would like to create a PCA biplot, this parameter should be the second component.
Z	a vector of numerics, the z coordinates of points in the plot. If user would like to create a PCA biplot, this parameter should be the third component.
class_label	a vector of characters, labels or categories of samples. The vector name should be sample names.
xlab	character, the label for x-axis. Default is "PC1".
ylab	character, the label for y-axis. Default is "PC2".
zlab	character, the label for z-axis. Default is "PC3".

legend_cex	numeric, giving the amount by which the text of legend should be magnified relative to the default. Default is 0.8.
main	character, an overall title for the plot. Default is "".
point_cex	numeric, giving the amount by which the size of the data points should be magnified relative to the default. Default is 1.
legend_pos	character, the position of legend. Default is "topright".
legend_ncol	integer, number of columns of legend. Default is 1.
use_color	a vector of color codes, colors to be assigned to each member of display label. Default is brewer.pal(9, 'Set1').
pre_define	a vector of characters, pre-defined color codes for a certain input (e.g. c("blue", "red") with names c("A", "B")). Default is NULL.
...	other paramters used in scatter3D.

Value

Return a logical value. If TRUE, the plot has been created successfully.

Return a logical value. If TRUE, the plot has been created successfully.

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 <- stats::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)
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 <- stats::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 *Heat Bubble Matrix Plot for Top Drivers in NetBID2 Analysis*

Description

draw.bubblePlot combines the matrix bubble chart and the heat map, using bubble color to compare P-values (performed by Fisher's Exact Test) and bubble size to compare the intersected size for target genes. Rows are enriched gene set, columns are top drivers. Users can also check number of protein-coding genes targetted by each driver.

draw.bubblePlot combines the matrix bubble chart and the heat map, using bubble color to compare P-values (performed by Fisher's Exact Test) and bubble size to compare the intersected size for target genes. Rows are enriched gene set, columns are top drivers. Users can also check number of protein-coding genes targetted by 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,
  only_return_mat = FALSE
)

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,
  only_return_mat = FALSE
)
```

Arguments

driver_list	a vector of characters, the names of top drivers.
show_label	a vector of characters, the names of top drivers to be displayed in the plot. If NULL, the names in driver_list will be displayed. Default is NULL.
Z_val	a vector of numerics, the Z statistics of the driver_list. It is highly suggested to assign names to this vector. If the vector is nameless, the function will use the names of driver_list by default.
driver_type	a vector of characters, the biotype or other characteristics of the driver. In the demo, we use "gene_biotype" column in the master table as input. It is highly suggested to assign names to this vector. If the vector is nameless, the function will use the names of driver_list by default. Default is NULL.
target_list	list, the driver-to-target list object. The names of the list elements are drivers. Each element is a data frame, usually contains at least three columns. "target", target gene names; "MI", mutual information; "spearman", spearman correlation coefficient. Users can call get_net2target_list to create this list and follow the suggested pipeline.
transfer2symbol2type	data.frame, the ID-conversion table for converting the original ID into gene symbol and gene biotype (at gene level), or into transcript symbol and transcript biotype (at transcript level). It is highly suggested to use get_IDtransfer2symbol2type to create this ID-conversion table.
bg_list	a vector of characters, a vector of background gene symbols. If NULL, genes in gs2gene will be used as background. Default is NULL.
min_gs_size	numeric, the minimum size of gene set to analysis. Default is 5.
max_gs_size	numeric, the maximum size of gene set to analysis, Default is 500.
gs2gene	list, a list contains elements of gene sets. The name of the element is gene set, each element contains a vector of genes in that gene set. If NULL, will use all_gs2gene, which is created by function gs.preload. Default is NULL.
use_gs	a vector of characters, the names of gene sets. If gs2gene is NULL, all_gs2gene will be used. And the use_gs must be the subset of names(all_gs2gene). Please check all_gs2gene_info for detailed cateogory description. Default is c("H", "CP:BIOCARTA", "CP:REACTOME", "CP:KEGG").
display_gs_list	a vector of characters, the names of gene sets to be displayed in the plot. If NULL, all the gene sets will be displayed in descending order of their significance. Default is NULL.
Pv_adj	character, method to adjust P-value. Default is "none". For details, please check p.adjust.methods.
Pv_thre	numeric, threshold for the adjusted P-values. Default is 0.1.
top_geneset_number	integer, the number of top enriched gene sets to be displayed in the plot. Default is 30.
top_driver_number	integer, the number of top significant drivers to be displayed in the plot. Default is 30.
pdf_file	character, the file path to save as PDF file. If NULL, no PDF file will be save. Default is NULL.

`main` character, an overall title for the plot.

`mark_gene` a vector of characters, a vector of gene symbols to be highlighted red in the plot. Default is NULL.

`driver_cex` numeric, giving the amount by which the text of driver symbols should be magnified relative to the default. Default is 1.

`gs_cex` numeric, giving the amount by which the text of gene set names should be magnified relative to the default. Default is 1.

`only_return_mat` logicial, if TRUE, the function will only return the gene set Vs. driver matrix with value representing the Z-statistics of the significance test; and the plot will not be generated. Default is FALSE.

Value

Return a logical value if only_return_mat=FALSE. If TRUE, the plot has been created successfully.

Return a logical value if only_return_mat=FALSE. If TRUE, the plot has been created successfully.

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 <- base::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$final_ms_tab
sig_driver <- draw.volcanoPlot(dat=ms_tab,label_col='gene_label',
                               logFC_col='logFC.G4.Vs.others_DA',
                               Pv_col='P.Value.G4.Vs.others_DA',
                               logFC_thre=0.4,
                               Pv_thre=1e-7,
                               main='Volcano Plot for G4.Vs.others_DA',
                               show_label=FALSE,
                               label_type = 'origin',
                               label_cex = 0.5)
gs.preload(use_spe='Homo sapiens',update=FALSE)
use_genes <- base::unique(analysis.par$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)
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 <- base::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 <- base::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` *Scatter Box Plot of Driver's Expression Values and Activity Values across Subgroup of Samples*

Description

draw.categoryValue draws a scatter box plot to visualize one selected driver's expression value and activity value across different phenotype subgroups of samples. Two side-by-side scatter box plots will be created. The left plot shows driver's activity values in different phenotype subgroups, each point is a sample. The right plot shows driver's expression value in different phenotype subgroups, each point is a sample.

draw.categoryValue draws a scatter box plot to visualize one selected driver's expression value and activity value across different phenotype subgroups of samples. Two side-by-side scatter box plots will be created. The left plot shows driver's activity values in different phenotype subgroups, each point is a sample. The right plot shows driver's expression value in different phenotype subgroups, each point is a sample.

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,
  use_color = NULL,
  pre_define = NULL
)

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,
  use_color = NULL,
  pre_define = NULL
)
```

Arguments

ac_val	a vector of numerics, the activity values of the selected driver across all samples.
exp_val	a vector of numerics, the expression values of the selected driver across all samples.
use_obs_class	a vector of characters, the category of sample. The order of samples here must match the order in ac_val and exp_val. Users can call get_obs_label to create this vector.
category_color	a vector of characters, a vector of color codes for each category in class_order. If NULL, will call get.class.color to create the vector. Default is NULL.
stripchart_color	character, the color of the scatter of points. Default is "black" with transparency alpha equals 0.7.
strip_cex	numeric, giving the amount by which the size of scattered points should be magnified relative to the default. Default is 1.
class_order	a vector of characters, the order of catefory (subgroup). If NULL, will use the alphabetical order of the category (subgroup). Default is NULL.
class_srt	numeric, rotation angle of the column labels (subgroup labels) at the bottom of the box plot. Default is 90.
class_cex	numeric, giving the amount by which the text of category (subgroup) labels should be magnified relative to the default. Default is 1.
pdf_file	character, the file path to save as PDF file. If NULL, no PDF file will be save. Default is NULL.
main_ac	character, the main title of the plot to show activity values. Default is "".
main_exp	character, the main title of the plot to show expression values. Default is "".
main_cex	numeric, giving the amount by which the text of the main title should be magnified relative to the default. Default is 1.
use_color	a vector of color codes, colors to be assigned to each member of display label. Default is brewer.pal(9, 'Set1').
pre_define	a vector of characters, pre-defined color codes for a certain input (e.g. c("blue", "red") with names c("A", "B")). Default is NULL.

Value

Return a logical value. If TRUE, the plot has been created successfully.

Return a logical value. If TRUE, the plot has been created successfully.

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

Description

`draw.clustComp` draws a table to show each sample's observed label vs. its predicted label. Each row represents an observed label (e.g. one subgroup of disease), each column represents the predicted label created by classification algorithm (e.g K-means).

`draw.clustComp` draws a table to show each sample's observed label vs. its predicted label. Each row represents an observed label (e.g. one subgroup of disease), each column represents the predicted label created by classification algorithm (e.g K-means).

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
)

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 labels created by classification (e.g K-means).
`obs_label`	a vector of characters, the observed labels annotated by phenotype data.
`strategy`	character, method to quantify the similarity between predicted labels vs. observed labels. Users can choose from "ARI (adjusted rand index)", "NMI (normalized mutual information)" and "Jaccard". Default is "ARI".
`use_col`	logical, If TRUE, the table will be colored. The more sample gathered in one table cell, the darker shade it has. Default is TRUE.

low_K	integer, a threshold of sample number to be shown in a single cell. If too many samples gathered in a single table cell, it will be challenging for eyes. By setting the value of this threshold, if the number of samples gathered in one table cell exceeded the threshold, only the number will be shown. Otherwise, all samples' names will be listed. Default is 5.
highlight_clust	a vector of characters, the predicted label need to be highlighted in the figure.
main	character, an overall title for the plot.
clust_cex	numeric, text size for the predicted label (column names). Default is 1.
outlier_cex	numeric, text size for the observed label (row names). Default is 0.3.

Details

The table provides more details about the side-by-side PCA biplot created by `draw.emb.kmeans`. The purpose is to find if any abnormal sample (outlier) exists. The darker the table cell is, the more samples are gathered in the corresponding label.

The table provides more details about the side-by-side PCA biplot created by `draw.emb.kmeans`. The purpose is to find if any abnormal sample (outlier) exists. The darker the table cell is, the more samples are gathered in the corresponding label.

Value

Return a matrix of integers and a table for visualization. Rows are predicted label, columns are observed label. Integer is the number of samples gathered in the corresponding label.

Return a matrix of integers and a table for visualization. Rows are predicted label, columns are observed label. Integer is the number of samples gathered in the corresponding label.

Examples

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

`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,
  display_col = "P.Value",
  z_col = "Z-statistics",
  digit_num = 2,
  row_cex = 1,
  column_cex = 1,
  text_cex = 1,
  pdf_file = NULL
)

draw.combineDE(
  DE_list = NULL,
  main_id = NULL,
  top_number = 30,
  display_col = "P.Value",
  z_col = "Z-statistics",
  digit_num = 2,
  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.
`display_col`	character, column names used to display. Default is 'P.Value'.
`z_col`	character, column names for Z statistics used for background color bar. Default is 'Z-statistics'.
`digit_num`	integer, number of digits to display on the plot. Default is 2.

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.

This plot function need to input the output from combineDE.

Value

logical value indicating whether the plot has been successfully generated

logical value indicating whether the plot has been successfully generated

Examples

```
analysis.par <- list()
analysis.par$out.dir.DATA <- system.file('demo1','driver/DATA/',package = "NetBID2")
NetBID.loadRDData(analysis.par=analysis.par,step='ms-tab')
phe_info <- Biobase::pData(analysis.par$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)
analysis.par <- list()
analysis.par$out.dir.DATA <- system.file('demo1','driver/DATA/',package = "NetBID2")
NetBID.loadRDData(analysis.par=analysis.par,step='ms-tab')
phe_info <- Biobase::pData(analysis.par$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.emb.kmeans` *Visualize the K-means Clustering Result by several dimension reduction and embedding methods.*

Description

`draw.emb.kmeans` is a data visualization function to show the K-means clustering result of a data matrix. A PCA/MDS/UMAP biplot is generated to visualize the clustering. Two biplots side-by-side will show the comparison between real observation labels (left) and the K-means predicted labels (right).

`draw.emb.kmeans` is a data visualization function to show the K-means clustering result of a data matrix. A PCA/MDS/UMAP biplot is generated to visualize the clustering. Two biplots side-by-side will show the comparison between real observation labels (left) and the K-means predicted labels (right).

Usage

```
draw.emb.kmeans(
  mat = NULL,
  embedding_method = "pca",
  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",
  main = "",
  verbose = TRUE,
  use_color = NULL,
  pre_define = NULL
)

draw.emb.kmeans(
  mat = NULL,
  embedding_method = "pca",
  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",
  main = "",
  verbose = TRUE,
  use_color = NULL,
  pre_define = NULL
)
```

Arguments

mat	a numeric data matrix, the columns (e.g. sample) will be clustered using the feature (e.g. genes) rows.
embedding_method	character, embedding method, choose from pca, mds and umap. Default is pca.
all_k	a vector of integers, a pre-defined K value. K is the number of final clusters. If NULL, the function will try all possible K values. Default is NULL.
obs_label	a vector of characters, a vector describes each sample's selected phenotype information, using sample name as vector name. Can be obtained by calling get_obs_label.
legend_pos	character, position of the plot legend. Default is 'topleft'.
legend_cex	numeric, text size of the plot legend. Default is 0.8.
plot_type	character, plot type. Users can choose from "2D", "2D.ellipse", "2D.interactive","2D.text" and "3D". Default is "2D.ellipse".
point_cex	numeric, size of the point in the plot. Default is 1.
kmeans_strategy	character, K-means clustering algorithm. Users can choose "basic" or "consensus". "consensus" is performed by ConsensusClusterPlus. Default is "basic".
choose_k_strategy	character, method to choose the K-value. Users can choose from "ARI (adjusted rand index)", "NMI (normalized mutual information)" and "Jaccard". Default is "ARI".
return_type	character, the type of result returned. Users can choose "optimal" or "all". "all", all the K-values in all_k will be returned. "optimal", only the K-value yielding the optimal classification result will be returned. Default is "optimal".
main	character, title for the plot.
verbose	logical, if TRUE, print out detailed information during calculation. Default is TRUE.
use_color	a vector of color codes, colors to be assigned to each member of display label. Default is brewer.pal(9, 'Set1').
pre_define	a vector of characters, pre-defined color codes for a certain input (e.g. c("blue", "red") with names c("A", "B")). Default is NULL.

Details

This function is mainly used to check the sample clustering result, in aim to detect if any abnormal (outlier) sample(s) exsist. The input is a high-throughput expression matrix. Each row is a gene/transcript/probe and each column is a sample. Users need to provide the real observation label for each sample. A K-value yielding the optimal classification result will be used to generate the

predicted labels. A comparision score (choose from ARI, NMI, Jaccard) will be calculated and shown in the figure.

This function is mainly used to check the sample clustering result, in aim to detect if any abnormal (outlier) sample(s) exsist. The input is a high-throughput expression matrix. Each row is a gene/transcript/probe and each column is a sample. Users need to provide the real observation label for each sample. A K-value yielding the optimal classification result will be used to generate the predicted labels. A comparision score (choose from ARI, NMI, Jaccard) will be calculated and shown in the figure.

Value

Return a vector of predicted labels, if `return_type` is set to "optimal". Or a list of all possible K-values, if `return_type` is set to be "all". If plot_type='2D.interactive', will return a plotly class object for interactive display.

Return a vector of predicted labels, if `return_type` is set to "optimal". Or a list of all possible K-values, if `return_type` is set to be "all". If plot_type='2D.interactive', will return a plotly class object for interactive display.

Examples

```
network.par <- list()
network.par$out.dir.DATA <- system.file('demo1','network/DATA/',package = "NetBID2")
NetBID.loadRDData(network.par=network.par,step='exp-QC')
mat <- Biobase::exprs(network.par$net.eset)
phe <- Biobase::pData(network.par$net.eset)
intgroup <- get_int_group(network.par$net.eset)
for(i in 1:base::length(intgroup)){
 print(intgroup[i])
 pred_label <- draw.emb.kmeans(mat=mat,all_k = NULL,obs_label=get_obs_label(phe,intgroup[i]))
 print(base::table(list(pred_label=pred_label,obs_label=get_obs_label(phe,intgroup[i]))))
}
pred_label <- draw.emb.kmeans(mat=mat,all_k = NULL,
                              obs_label=get_obs_label(phe,'subgroup'),
                              kmeans_strategy='consensus')
## interactive display
draw.emb.kmeans(mat=mat,all_k = NULL,
                obs_label=get_obs_label(phe,'subgroup'),
                plot_type='2D.interactive',
                pre_define=c('WNT'='blue','SHH'='red','G4'='green'))
network.par <- list()
network.par$out.dir.DATA <- system.file('demo1','network/DATA/',package = "NetBID2")
NetBID.loadRDData(network.par=network.par,step='exp-QC')
mat <- Biobase::exprs(network.par$net.eset)
phe <- Biobase::pData(network.par$net.eset)
intgroup <- get_int_group(network.par$net.eset)
for(i in 1:base::length(intgroup)){
 print(intgroup[i])
 pred_label <- draw.emb.kmeans(mat=mat,all_k = NULL,obs_label=get_obs_label(phe,intgroup[i]))
 print(base::table(list(pred_label=pred_label,obs_label=get_obs_label(phe,intgroup[i]))))
}
pred_label <- draw.emb.kmeans(mat=mat,all_k = NULL,
                              obs_label=get_obs_label(phe,'subgroup'),
                              kmeans_strategy='consensus')
## interactive display
draw.emb.kmeans(mat=mat,all_k = NULL,
```

```
obs_label=get_obs_label(phe,'subgroup'),
plot_type='2D.interactive',
pre_define=c('WNT'='blue','SHH'='red','G4'='green'))
```

`draw.eset.QC` *QC plots for ExpressionSet class object.*

Description

`draw.eset.QC` is a function to draw a set of plots for quality control analysis. 6 types of plots will be created, including heatmap, dimension reduction plots (pca, mds, umap), boxplot, density, correlation and meansd.

`draw.eset.QC` is a function to draw a set of plots for quality control analysis. 6 types of plots will be created, including heatmap, dimension reduction plots (pca, mds, umap), boxplot, density, correlation and meansd.

Usage

```
draw.eset.QC(
  eset,
  outdir = ".",
  do.logtransform = FALSE,
  intgroup = NULL,
  prefix = "",
  choose_plot = c("heatmap", "pca", "boxplot", "density", "correlation"),
  generate_html = TRUE,
  correlation_strategy = "pearson",
  plot_all_point = FALSE,
  emb_plot_type = "2D.ellipse",
  use_color = NULL,
  pre_define = NULL
)

draw.eset.QC(
  eset,
  outdir = ".",
  do.logtransform = FALSE,
  intgroup = NULL,
  prefix = "",
  choose_plot = c("heatmap", "pca", "boxplot", "density", "correlation"),
  generate_html = TRUE,
  correlation_strategy = "pearson",
  plot_all_point = FALSE,
  emb_plot_type = "2D.ellipse",
  use_color = NULL,
  pre_define = NULL
)
```

Arguments

eset	ExpressionSet class, quality control analysis target.
outdir	character, the directory path for saving output files.
do.logtransform	logical, if TRUE, the log transformation will be performed on the gene expression value. Default is FALSE.
intgroup	a vector of characters, the interested phenotype groups from the ExpressionSet. If NULL, it will automatcially extract all possible groups by get_int_group. Default is NULL.
prefix	character, the prefix for the QC figures' name. Default is "".
choose_plot	a vector of characters, choose one or many from 'heatmap', 'pca','mds','umap','boxplot', 'density','correlation' and 'meansd' plots. Default is 'heatmap', 'pca', 'boxplot','density' and 'correlation'
generate_html	logical, if TRUE, it will generate a html file by R Markdown. Otherwise, it will generate separate PDF files. Default is TRUE.
correlation_strategy	character, the strategy to calculate the sample correlation, choose from 'pearson' and 'spearman'. Default is 'pearson'.
plot_all_point	logical, if TRUE, the scatterplot will plot all points in the correlation, otherwise will plot the main trend for reducing the figure size. Default is FALSE.
emb_plot_type	character, plot type for dimension reduction methods (pca, mds, umap). Users can choose from "2D","2D.interactive", "2D.ellipse", "2D.text" and "3D". Default is "2D.ellipse".
use_color	a vector of color codes, colors to be assigned to each member of display label. Default is brewer.pal(9, 'Set1').
pre_define	a vector of characters, pre-defined color codes for a certain input (e.g. c("blue", "red") with names c("A", "B")). Default is NULL.

Examples

```
## Not run:
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)
network.par$out.dir.QC <- getwd() ## set the output directory
draw.eset.QC(network.par$net.eset,outdir=network.par$out.dir.QC,intgroup=intgroups,
             pre_define=c('WNT'='blue','SHH'='red','G4'='green'))
draw.eset.QC(network.par$net.eset,outdir=network.par$out.dir.QC,intgroup=intgroups,
             pre_define=c('WNT'='blue','SHH'='red','G4'='green'),
             emb_plot_type = '2D.interactive',choose_plot = 'pca')

## End(Not run)
## Not run:
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)
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,
             pre_define=c('WNT'='blue','SHH'='red','G4'='green'))
```

```
draw.eset.QC(network.par$net.eset,outdir=network.par$out.dir.QC,intgroup=intgroups,
             pre_define=c('WNT'='blue','SHH'='red','G4'='green'),
             emb_plot_type = '2D.interactive',choose_plot = 'pca')

## End(Not run)
```

`draw.funcEnrich.bar` *Bar Plot for Gene Set Enrichment Analysis Result*

Description

`draw.funcEnrich.bar` draws a horizontal bar plot to visualize the gene set enrichment analysis. Users can choose to display P-values and the top intersected genes from each gene set.

`draw.funcEnrich.bar` draws a horizontal bar plot to visualize the gene set enrichment analysis. Users can choose to display P-values and the top intersected genes from each gene set.

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
)

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 of functional enrichment analysis. It is highly suggested to use funcEnrich.Fisher to create this data frame. If users decided to prepare the data.frame on their own, please make sure the column names match the following parameters.
top_number	numeric, the number of top enriched gene sets to be displayed. Default is 30.
Pv_col	character, the name of the column in funcEnrich_res which contains P-value. Default is "Ori_P".
item_col	character, the name of the column in funcEnrich_res which contains intersected genes collapsed with ";". Default is "Intersected_items".
Pv_thre	numeric, threshold of P-values. Genes or drivers with P-values lower than the threshold will be kept. Default is 0.1.
display_genes	logical, if TRUE, the intersected genes will be displayed. Default is FALSE.
name_col	character, the name of the column in funcEnrich_res which contains gene set name. Default is "#Name".
gs_cex	numeric, giving the amount by which the text of gene sets names should be magnified relative to the default. Default is 0.5.
gene_cex	numeric, giving the amount by which the text of gene symbols should be magnified relative to the default. Default is 0.5.
main	character, an overall title for the plot.
bar_col	character, the color code used to plot the bars. Default is brewer.pal(8,'RdBu')[7].
eg_num	numeric, the number of intersected gene symbols to display on the right side of the bar. Default is 5.
pdf_file	character, the file path to save as PDF file. If NULL, no PDF file will be saved. Default is NULL.

Value

Return a logical value. If TRUE, the plot has been created successfully.

Return a logical value. If TRUE, the plot has been created successfully.

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=3,
                  main='Function Enrichment for Top drivers',
                  display_genes = TRUE,eg_num=3,
                  gs_cex=0.3,gene_cex=0.3)
## 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)
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=3,
                    main='Function Enrichment for Top drivers',
                    display_genes = TRUE,eg_num=3,
                    gs_cex=0.3,gene_cex=0.3)
## 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 Gene Set Enrichment Analysis Result*

Description

`draw.funcEnrich.cluster` draws a cluster plot based on binary matrix, to visualize the existence of genes in the enriched gene sets. The P-value of enrichment is also displayed in the plot.

`draw.funcEnrich.cluster` draws a cluster plot based on binary matrix, to visualize the existence of genes in the enriched gene sets. The P-value of enrichment is also displayed in the plot.

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,
  inner_color = brewer.pal(9, "Reds")[3],
  cluster_gs = TRUE,
  cluster_gene = TRUE,
  pdf_file = NULL,
  use_genes = NULL,
  return_mat = FALSE
)

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,
  inner_color = brewer.pal(9, "Reds")[3],
  cluster_gs = TRUE,
  cluster_gene = TRUE,
  pdf_file = NULL,
  use_genes = NULL,
  return_mat = FALSE
)
```

Arguments

funcEnrich_res	data.frame, containing the result of functional enrichment analysis. It is highly suggested to use funcEnrich.Fisher to create this data frame. If users decided to prepare the data.frame on their own, please make sure the column names match the following parameters.
top_number	numeric, the number of top enriched gene sets to be displayed. Default is 30.
Pv_col	character, the name of the column in funcEnrich_res which contains P-value. Default is "Ori_P".
name_col	character, the name of the column in funcEnrich_res which contains gene set name. Default is "#Name".
item_col	character, the name of the column in funcEnrich_res which contains intersected genes collapsed with ";". Default is "Intersected_items".
Pv_thre	numeric, threshold of P-values. Genes or drivers with P-values lower than the threshold will be kept. Default is 0.1.

gs_cex	numeric, giving the amount by which the text of gene sets names should be magnified relative to the default. Default is 0.5.
gene_cex	numeric, giving the amount by which the text of gene symbols should be magnified relative to the default. Default is 0.8.
pv_cex	numeric, giving the amount by which the text of P-values should be magnified relative to the default. Default is 0.7.
main	character, an overall title for the plot.
h	numeric, the height where the cluster tree should be cut. The same parameter as cutree. Default is 0.95.
inner_color	character, the color code for the indexing box inside the main plot region. Could be one character or a character vector. If want to set the color by gene, could input the color code character with names set as genes. If want to set the color by Z-score, could use 'z2col' to generate the color code. Default is brewer.pal(9,'Reds')[3].
cluster_gs	logical, if TRUE, gene sets will be clustered. Default is TRUE.
cluster_gene	logical, if TRUE, genes will be clustered. Default is TRUE.
pdf_file	character, the file path to save as PDF file. If NULL, no PDF file will be saved. Default is NULL.
use_genes	a vector of characters, a vector of gene symbols to display. If NULL, all the genes in the top enriched gene sets will be displayed. Default is NULL.
return_mat	logical, if TRUE, return a binary matrix. Rows are gene sets, columns are genes. Default if FALSE.

Value

If return_mat==FALSE, return a logical value. If TRUE, plot has been created successfully. If return_mat == TRUE, return a binary matrix of the cluster. Rows are gene sets, columns are genes.

If return_mat==FALSE, return a logical value. If TRUE, plot has been created successfully. If return_mat == TRUE, return a binary matrix of the cluster. Rows are gene sets, columns are genes.

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)
DA_Z <- z2col(ms_tab[rownames(sig_driver),'Z.G4.Vs.others_DA'],
```

```
        blue_col=brewer.pal(9,'Blues')[3],
        red_col=brewer.pal(9,'Reds')[3],col_max_thre=6)
names(DA_Z) <- ms_tab[rownames(sig_driver),'geneSymbol']
draw.funcEnrich.cluster(funcEnrich_res=res1,top_number=30,gs_cex = 0.5,
                        gene_cex=0.9,pv_cex=0.8,inner_color=DA_Z)
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$out.dir.DATA <- system.file('demo1','driver/DATA/',package = "NetBID2")
NetBID.loadRData(analysis.par=analysis.par,step='ms-tab')
ms_tab <- analysis.par$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$out.dir.PLOT <- getwd() ## directory for saving the pdf files
draw.funcEnrich.cluster(funcEnrich_res=res1,top_number=30,gs_cex = 0.8,
                        gene_cex=0.9,pv_cex=0.8,
                        pdf_file = sprintf('%s/funcEnrich_cluster.pdf',
                        analysis.par$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)
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')
draw.funcEnrich.cluster(funcEnrich_res=res1,top_number=30,gs_cex = 0.5,
                        gene_cex=0.9,pv_cex=0.8)
DA_Z <- z2col(ms_tab[rownames(sig_driver),'Z.G4.Vs.others_DA'],
        blue_col=brewer.pal(9,'Blues')[3],
        red_col=brewer.pal(9,'Reds')[3],col_max_thre=6)
names(DA_Z) <- ms_tab[rownames(sig_driver),'geneSymbol']
draw.funcEnrich.cluster(funcEnrich_res=res1,top_number=30,gs_cex = 0.5,
                        gene_cex=0.9,pv_cex=0.8,inner_color=DA_Z)
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 one Gene Set or one Driver*

Description

`draw.GSEA` draws a GSEA plot to analyze one gene set (with gene list annotated) or one driver (with list of target genes).

`draw.GSEA` draws a GSEA plot to analyze one gene set (with gene list annotated) or one driver (with list of target genes).

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
)

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 named vector of numerics, the differential values (DE or DA) calculated from a sample comparison (e.g. "G4 vs. Others"). Names of the vector must be gene names. For the DA, user could use 'processDriverProfile()' to convert the DA profile into gene-name based profile. The differential values can be "logFC" or "t-statistics".
use_genes	a vector of characters, a vector of genes to display. The genes can either be annotated genes in gene set or the targe genes from a specific driver. The gene names must be a subset of `names(rank_profile)`.
use_direction	a vector of numeric 1s and -1s, 1 is positive regulation from driver, -1 is negative regulation from driver. Users can get this vector by converting the signs of "spearman". If NULL, no regulation direction will be displayed. Default is NULL.
main	character, an overall title for the plot. Default is "".
pdf_file	character, the file path to save as PDF file. If NULL, no PDF file will be saved. Default is NULL.
annotation	character, the annotation set by users for easier reference. Normally the annotation is the P-value or other statistics to show the significance of the interested gene set or driver. If NULL, will perform a Kolmogorov-Smirnov test to get the significance value. If want to get the statistics from GSEA test, could use 'funcEnrich.GSEA()' to get the statistics first. Default is NULL.
annotation_cex	numeric, giving the amount by which the text of annotation should be magnified relative to the default. 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 "".

Value

Return a logical value. If TRUE, the plot has been created successfully.

Return a logical value. If TRUE, the plot has been created successfully.

Examples

```
gs.preload(use_spe='Homo sapiens',update=FALSE)
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

## draw for the most significant gene set
# by driver's DA profile
DA_profile <- processDriverProfile(Driver_name=ms_tab$gene_label,
                                   Driver_profile=ms_tab$logFC.G4.Vs.others_DA,
                                   choose_strategy='absmax',
                                   return_type ='gene_statistics')
res1 <- funcEnrich.GSEA(rank_profile=DA_profile,
                        use_gs=c('H'),
                        Pv_thre=0.1,Pv_adj = 'none')
top_gs <- res1[1,'#Name'] ## draw for the top 1
annot <- sprintf('NES: %s \nAdjusted P-value: %s',
          signif(res1[1,'NES'],2),
          signif(res1[1,'Adj_P'],2))
draw.GSEA(rank_profile=DA_profile,
          use_genes=all_gs2gene$H[[top_gs]],
          main=sprintf('GSEA plot for gene set %s',
          top_gs),
          annotation=annot,annotation_cex=1.2,
          left_annotation='high in G4',
          right_annotation='high in others')

## draw for the most significant driver
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)
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

## draw for the most significant gene set
# by driver's DA profile
DA_profile <- processDriverProfile(Driver_name=ms_tab$gene_label,
                                   Driver_profile=ms_tab$logFC.G4.Vs.others_DA,
                                   choose_strategy='absmax',
                                   return_type ='gene_statistics')
res1 <- funcEnrich.GSEA(rank_profile=DA_profile,
                        use_gs=c('H'),
                        Pv_thre=0.1,Pv_adj = 'none')
top_gs <- res1[1,'#Name'] ## draw for the top 1
annot <- sprintf('NES: %s \nAdjusted P-value: %s',
          signif(res1[1,'NES'],2),
          signif(res1[1,'Adj_P'],2))
draw.GSEA(rank_profile=DA_profile,
          use_genes=all_gs2gene$H[[top_gs]],
          main=sprintf('GSEA plot for gene set %s',
          top_gs),
          annotation=annot,annotation_cex=1.2,
          left_annotation='high in G4',
          right_annotation='high in others')

## draw for the most significant driver
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$out.dir.DATA <- system.file('demo1','driver/DATA/',package = "NetBID2")
NetBID.loadRData(analysis.par=analysis.par,step='ms-tab')
ms_tab <- analysis.par$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)
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` *Draw GSEA (gene set enrichment analysis) Plot with NetBID Analysis of Drivers*

Description

`draw.GSEA.NetBID` creates a GSEA plot for drivers with more NetBID analysis information. Such as number of target genes, ranking of target genes in differential expressed file, differential expression (DE) and differential activity (DA) values.

`draw.GSEA.NetBID` creates a GSEA plot for drivers with more NetBID analysis information. Such as number of target genes, ranking of target genes in differential expressed file, differential expression (DE) and differential activity (DA) values.

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
)

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, a data.frame created either by function getDE.limma.2G or getDE.BID.2G. Row names are gene/driver names, columns must include gene/driver name and calculated differencial values (e.g. "ID", "logFC", "AveExpr", "P.Value" etc.).
name_col	character, the name of the column in DE contains gene names. If NULL, will use the row names of DE. Default is NULL.
profile_col	character, the name of the column in DE contains calculated differencial value (e.g. "logFC" or "P.Value"). If DE is created by getDE.limma.2G or getDE.BID.2G, this parameter should be set to "logFC" or "t".
profile_trend	character, users can choose between "pos2neg" and "neg2pos". "pos2neg" means high profile_col in target group will be shown on the left. "neg2pos" means high profile_col in control group will be shown on the left. Default is "pos2neg". For details, please check online tutorial.
driver_list	a vector of characters, the names of top drivers.
show_label	a vector of characters, the names of top drivers. If NULL, will display the names in driver_list. Default is NULL.
driver_DA_Z	a vector of numerics, the Z statistics of differential activity (DA) value of the driver_list. It is highly suggested to give names to the vector, otherwise the names of driver_list will be used.
driver_DE_Z	a vector of numerics, the Z statistics of differential expressed (DE) value of the driver_list. It is highly suggested to give names to the vector, otherwise the names of driver_list will be used.
target_list	list, the driver-to-target list object. The names of the list elements are drivers. Each element is a data frame, usually contains at least three columns. "target", target gene names; "MI", mutual information; "spearman", spearman correlation coef- ficient. Users can call get_net2target_list to create this list.
top_driver_number	numeric, number for the top significant drivers to be displayed in the plot. Default is 30.
target_nrow	numeric, users can choose between 1 and 2. Number of panels to mark the ranking of target genes. If 1, the ranking of target genes will be marked in one panel. If 2, the ranking of target genes will be marked in two panels. Upper panel for positively-regulated, lower panel for negatively-regulated. Default is 2. For details, please check online tutorial.
target_col	character, name of the color palette used for display marker line in the panel. Users can choose between "black" and "RdBu". If "black", the marker line in the panel is black. If "RdBu", the marker line in the panel is Red to Blue. 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, name of the color palette used for display target genes. This parameter works only when `target_col` is set as "RdBu". Users can choose between "PN" and "DE". If "PN", positively-regulated genes will be colored red and negatively-regulated genes will be colored blue. If "DE", the color shades is decided by its differentiated value. Default is "PN".

`left_annotation`
: character, annotation on the left of profile curve, indicating high in control group or target group. Default is "".

`right_annotation`
: character, annotation on the right of profile curve, indicating high in the opposite group of `left_annotation`. Default is "".

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

`profile_sig_thre`
: numeric, threshold value for target genes. This parameter works only when `target_col_type` is set as "DE" and `target_col` is set as "RdBu". Non-significant target genes will be colored grey. Default is 0.

`Z_sig_thre`
: numeric, threshold value of Z statistics from `driver_DA_Z` and `driver_DE_Z`. Significant values will have background color. Default is 1.64.

`pdf_file`
: character, the file path to save figure as PDF file. If NULL, no PDF file will be saved. Default is NULL.

Value

Return a logical value. If TRUE, the plot has been created successfully.

Return a logical value. If TRUE, the plot has been created successfully.

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
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)
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.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)
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` *Draw GSEA (gene set enrichment analysis) Plot with NetBID Analysis of Gene Sets*

Description

`draw.GSEA.NetBID.GS` creates a GSEA plot for gene sets with more NetBID analysis information. Such as, number of genes in each gene set, marking the rank of annotated genes in the differential expression profile and differential activity (DA) values.

`draw.GSEA.NetBID.GS` creates a GSEA plot for gene sets with more NetBID analysis information. Such as, number of genes in each gene set, marking the rank of annotated genes in the differential expression profile and differential activity (DA) values.

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
)

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, a data.frame created either by function getDE.limma.2G or getDE.BID.2G. Row names are gene names, columns must include the calculated differencial values (e.g. "ID", "logFC", "AveExpr", "P.Value" etc.).
name_col	character, the name of the column in DE contains gene names. If NULL, will use the row names of DE. Default is NULL.
profile_col	character, the name of the column in DE contains calculated differencial value (e.g. "logFC" or "P.Value"). If DE is created by getDE.limma.2G or getDE.BID.2G, this parameter should be set to "logFC" or "t".
profile_trend	character, users can choose between "pos2neg" and "neg2pos". "pos2neg" means high profile_col in target group will be shown on the left. "neg2pos" means high profile_col in control group will be shown on the left. Default is "pos2neg".
sig_gs_list	a vector of characters, the names of top gene sets.
gs_DA_Z	a vector of numerics, the Z-statistics of differentail activity (DA) values for the sig_gs_list. It is highly suggested to assign name to the vector, otherwise will use name of sig_gs_list.
use_gs2gene	list, contains elements of gene sets. Element name is gene set name, each element contains a vector of genes belong to that gene set. This list can be created by calling one element from all_gs2gene, or merge several gene sets into one by using merge_gs.

top_gs_number integer, the number of top significant gene sets to be displayed. Default is 30.

target_col character, name of the color palette used for display marker line in the panel. Users can choose between "black" and "RdBu". If "black", the marker line in the panel is black. If "RdBu", the marker line in the panel is Red to Blue. The color shade of the marker line is decided by each gene's significance of differentiation. High in red, low in blue. Default is "RdBu".

left_annotation character, annotation on the left of profile curve, indicating high in control group or target group. Default is "".

right_annotation character, annotation on the right of profile curve, indicating high in the opposite group of left_annotation. Default is "".

main character, an overall title for the plot. Default is "".

profile_sig_thre numeric, threshold value for target genes. This parameter works only when target_col_type is set as "DE" and target_col is set as "RdBu". Non-significant target genes will be colored grey. Default is 0.

Z_sig_thre numeric, threshold value of Z-statistics from driver_DA_Z and driver_DE_Z. Significant values will have background color. Default is 1.64.

pdf_file character, the file path to save figure as PDF file. If NULL, no PDF file will be saved. Default is NULL.

Value

Return a logical value. If TRUE, the plot has been created successfully.

Return a logical value. If TRUE, the plot has been created successfully.

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('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]]
analysis.par$out.dir.PLOT <- getwd()

## directory for saving the pdf files
exp_mat_gene <- Biobase::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 <- Biobase::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$out.dir.PLOT))
## GSEA plot for the significant gene sets
draw.GSEA.NetBID.GS(DE=DE,name_col='ID',
                    profile_col='t',profile_trend='pos2neg',
                    sig_gs_list = sig_gs$ID,
                    gs_DA_Z=DA_gs[sig_gs$ID,'Z-statistics'],
                    use_gs2gene = use_gs2gene,
                    top_gs_number=5,target_col='RdBu',
                    left_annotation = 'test_left',
                    right_annotation = 'test_right',
                    main='test',Z_sig_thre=1.64,profile_sig_thre = 0,
                    pdf_file=sprintf('%s/NetBID_GSEA_GS_demo1.pdf',
                    analysis.par$out.dir.PLOT))

## End(Not run)
## 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('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]]
analysis.par$out.dir.PLOT <- getwd()

## directory for saving the pdf files
exp_mat_gene <- Biobase::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 <- Biobase::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$ID,
                    gs_DA_Z=DA_gs[sig_gs$ID,'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` *Draw Heatmap Plot to Display the Expression Level or Activity Level of Genes and Drivers*

Description

`draw.heatmap` plots the heatmap to see the expression level or activity level of genes and drivers across selected samples.

`draw.heatmap` plots the heatmap to see the expression level or activity level of 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",
  use_color = NULL,
  pre_define = NULL,
  ...
)

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",
  use_color = NULL,
  pre_define = NULL,
  ...
)
```

Arguments

mat	numeric matrix, the expression/activity matrix. Rows are genes or drivers, columns are selected samples.
use_genes	a vector of characters, selected genes (e.g. "originID"). Default is row names of mat.
use_gene_label	a vector of characters, a vector of labels for use_genes (e.g. "geneSymbol" or "gene_label"). Default is use_genes.
use_samples	a vector of characters, selected samples. Default is column names of mat.
use_sample_label	a vector of characters, a vector of labels for use_samples. Default is use_samples.

phenotype_info data.frame, phenotype of samples. Users can call Biobase::pData(eset) to create. The row names should match the column names in mat. Default is NULL.

use_phe a list of characters, selected phenotype of samples. A subset of columns from phenotype_info. Default is NULL.

main character, an overall title for the plot. Default is "".

scale character, users can choose from "row", "column" and "none". Indicating if the values should be centered and scaled in either the row direction or the column direction, or none. Default is "none".

pdf_file character, the file path to save plot as PDF file. If NULL, no PDF file will be saved. Default is NULL.

cluster_rows, cluster_columns
logical, the same parameters in Heatmap. Please check ?Heatmap for more details. Default is TRUE.

show_row_names, show_column_names
logical, the same parameters used in Heatmap. Please check ?Heatmap for more details. Default is TRUE.

clustering_distance_rows, clustering_distance_columns
character, the same parameters used in Heatmap. Please check ?Heatmap for more details. Default is "pearson".

use_color a vector of color codes, colors to be assigned to each member of display label. Default is brewer.pal(9, 'Set1').

pre_define a vector of characters, pre-defined color codes for a certain input (e.g. c("blue", "red") with names c("A", "B")). Default is NULL.

..., for more options, please check ?Heatmap for more details.

Value

Return a logical value. If TRUE, the plot has been created successfully.

Return a logical value. If TRUE, the plot has been created successfully.

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 <- Biobase::exprs(analysis.par$cal.eset) ## rownames matches originalID
ac_mat <- Biobase::exprs(analysis.par$merge.ac.eset) ## rownames matches originalID_label
phe_info <- Biobase::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),
            pre_define=c('WNT'='blue','SHH'='red','G4'='green'))
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(ac_mat),
             use_sample_label=phe_info[colnames(ac_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),
             pre_define=c('WNT'='blue','SHH'='red','G4'='green'))

## 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 <- Biobase::exprs(analysis.par$cal.eset) ## rownames matches originalID
ac_mat <- Biobase::exprs(analysis.par$merge.ac.eset) ## rownames matches originalID_label
phe_info <- Biobase::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),
            pre_define=c('WNT'='blue','SHH'='red','G4'='green'))
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(ac_mat),
             use_sample_label=phe_info[colnames(ac_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),
            pre_define=c('WNT'='blue','SHH'='red','G4'='green'))

## End(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 <- Biobase::exprs(analysis.par$cal.eset) ## rownames matches originalID
ac_mat <- Biobase::exprs(analysis.par$merge.ac.eset) ## rownames matches originalID_label
phe_info <- Biobase::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),
            pre_define=c('WNT'='blue','SHH'='red','G4'='green'))
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(ac_mat),
             use_sample_label=phe_info[colnames(ac_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),
             pre_define=c('WNT'='blue','SHH'='red','G4'='green'))

## 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 <- Biobase::exprs(analysis.par$cal.eset) ## rownames matches originalID
ac_mat <- Biobase::exprs(analysis.par$merge.ac.eset) ## rownames matches originalID_label
phe_info <- Biobase::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$out.dir.PLOT),
             pre_define=c('WNT'='blue','SHH'='red','G4'='green'))
draw.heatmap(mat=ac_mat,use_genes=ms_tab[rownames(sig_driver),'originalID_label'],
              use_gene_label=ms_tab[rownames(sig_driver),'gene_label'],
              use_samples=colnames(ac_mat),
              use_sample_label=phe_info[colnames(ac_mat),'geo_accession'],
              phenotype_info=phe_info,
              use_phe=c('gender','pathology','subgroup'),
              main='Activity for Top drivers',scale='row',
              cluster_rows=TRUE,cluster_columns=TRUE,
              clustering_distance_rows='pearson',
              clustering_distance_columns='pearson',
              row_names_gp = gpar(fontsize = 6),
              pdf_file=sprintf('%s/heatmap_demo2.pdf',
              analysis.par$out.dir.PLOT),
              pre_define=c('WNT'='blue','SHH'='red','G4'='green'))

## End(Not run)
```

`draw.MICA` *Draw Cluster Plot Using MICA (cluster algorithm)*

Description

`draw.MICA` is a function to visualize the cluster result for the samples using MICA (mutual information based clustering analysis) algorithm. Users need to give the MICA project information (directory and name), and the samples real labels. MICA returns the K-value that yields the best clustering performance. Users can pick one comparison score to show in the plot, "ARI", "NMI" or "Jaccard". MICA is not suggested, when sample size is small. This function may not work well for the updated version of MICA.

draw.MICA is a function to visualize the cluster result for the samples using MICA (mutual information based clustering analysis) algorithm. Users need to give the MICA project information (directory and name), and the samples real labels. MICA returns the K-value that yields the best clustering performance. Users can pick one comparison score to show in the plot, "ARI", "NMI" or "Jaccard". MICA is not suggested, when sample size is small. This function may not work well for the updated version of MICA.

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",
  main = "",
  verbose = TRUE,
  use_color = NULL,
  pre_define = NULL
)

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",
  main = "",
  verbose = TRUE,
  use_color = NULL,
  pre_define = NULL
)
```

Arguments

outdir	character, the output directory for running MICA.
prjname	charater, the project name for running MICA.
all_k	a vector of integers, the pre-defined K-values. If NULL, will use all possible K. Default is NULL.
obs_label	a vector of characters, the observed sample labels or categories.

legend_pos	character, position of the legend in the plot. Default is "topleft".
legend_cex	numeric, giving the amount by which the text of legend should be magnified relative to the default. Default is 0.8.
point_cex	numeric, giving the amount by which the size of the data points should be magnified relative to the default. Default is 1.
plot_type	character, type of the plot. Users can choose from "2D", "2D.ellipse", "2D.text" and "3D". Default is "2D.ellipse".
choose_k_strategy	character, method to choose the K-value. Users can choose from "ARI (adjusted rand index)", "NMI (normalized mutual information)" and "Jaccard". Default is "ARI".
visualization_type	character, users can choose from "tsne", "umap" and "mds". Default is "tsne".
return_type	character, the type of result returned. Users can choose "optimal" or "all". "all", all the K-values in all_k will be returned. "optimal", only the K-value yielding the optimal classification result will be returned. Default is "optimal".
main	character, title for the plot.
verbose	logical, if TRUE, print out detailed information during calculation. Default is TRUE.
use_color	a vector of color codes, colors to be assigned to each member of display label. Default is brewer.pal(9, 'Set1').
pre_define	a vector of characters, pre-defined color codes for a certain input (e.g. c("blue", "red") with names c("A", "B")). Default is NULL.

Value

Return a vector of the predicted label (if return_type is "optimal") and a list of all possible K- values (if return_type is "all").

Return a vector of the predicted label (if return_type is "optimal") and a list of all possible K- values (if return_type is "all").

---

draw.NetBID *Create Plot to show Differential Expression and Differential Activity Analysis of Top Drivers*

Description

draw.NetBID creates two side-by-side heatmaps to show the result of NetBID analysis. Both the differential expression analysis (the right heatmap) and differential activity analysis (the left heatmap) of top drivers are shown in the plot.

draw.NetBID creates two side-by-side heatmaps to show the result of NetBID analysis. Both the differential expression analysis (the right heatmap) and differential activity analysis (the left heatmap) of top drivers are shown in the plot.

Usage

```
draw.NetBID(
  DA_list = NULL,
  DE_list = NULL,
  main_id = NULL,
  top_number = 30,
  DA_display_col = "P.Value",
  DE_display_col = "logFC",
  z_col = "Z-statistics",
  digit_num = 2,
  row_cex = 1,
  column_cex = 1,
  text_cex = 1,
  col_srt = 60,
  pdf_file = NULL
)

draw.NetBID(
  DA_list = NULL,
  DE_list = NULL,
  main_id = NULL,
  top_number = 30,
  DA_display_col = "P.Value",
  DE_display_col = "logFC",
  z_col = "Z-statistics",
  digit_num = 2,
  row_cex = 1,
  column_cex = 1,
  text_cex = 1,
  col_srt = 60,
  pdf_file = NULL
)
```

Arguments

DA_list	list, contains the differential activity (DA) analysis result.
DE_list	list, contains the differential expression (DE) analysis result.
main_id	character, the top genes/drivers from which DA comparison group. Must be one of the names in DA_list. If NULL, the first element name of DA_list will be used. Default is NULL.
top_number	integer, the number of the top significant genes/drivers to be displayed in the plot. Default is 30.
DA_display_col	character, which statistic column from NetBID analysis is to be used as the column of the left heatmap. Default is "P.Value".
DE_display_col	character, which statistic column from NetBID analysis is to be used as the column of the right heatmap. Default is "logFC".
z_col	character, which statistic column from NetBID analysis is to be used as the color scale of the heatmap. Default is "Z-statistics".
digit_num	integer, indicating the number of decimal places (round) or significant digits (signif) to be used. Default is 2.

row_cex	numeric, giving the amount by which the text of row names should be magnified relative to the default. Default is 1.
column_cex	numeric, giving the amount by which the text of column names should be maginified relative to the default. Default is 1.
text_cex	numeric, giving the amount by which the text of in the table cell should be maginified relative to the default. Default is 1.
col_srt	numeric, rotation angle of the column labels at the bottom of heatmap. Default is 60.
pdf_file	character, the path to save the plot as PDF file. If NULL, PDF file will not be generated. Default is NULL.

Value

Return a logical value. If TRUE, the plot is created successfully.

Return a logical value. If TRUE, the plot is created successfully.

Examples

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

---

draw.network.QC *QC Tables and Plots for Network Object*

Description

draw.network.QC creates tables and plots for showing some basic statistics, driver statistics and scale-free checking of the target network.

draw.network.QC creates tables and plots for showing some basic statistics, driver statistics and scale-free checking of the target network.

Usage

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

draw.network.QC(
```

```
  igraph_obj,
  outdir = NULL,
  prefix = "",
  directed = TRUE,
  weighted = FALSE,
  generate_html = TRUE,
  html_info_limit = TRUE
)
```

Arguments

igraph_obj	igraph object, created by get.SJAracne.network.
outdir	character, the output directory for saving QC tables and plots.
prefix	character, the prefix of output QC figures names. Default is "".
directed	logical, if TRUE, this network will be treated as a directed network. Default is TRUE.
weighted	logical, if TRUE, this network will be treated as a weighted network. Default is FALSE.
generate_html	logical, if TRUE, a html file will be created by R Markdown. If FALSE, plots will be save as separated PDFs. Default is TRUE.
html_info_limit	logical, if TRUE, the statistics for network QC html will be limited. Default is TRUE.

Value

Return a logical value. If TRUE, success in creating QC tables and plots.

Return a logical value. If TRUE, success in creating QC tables and plots.

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,
                                            project_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)
## 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,
                                            project_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.oncoprint` *Draw Oncoprint Plot to display the mutation information*

Description

`draw.oncoprint` plots the heatmap to display the mutation information for samples.

`draw.oncoprint` plots the heatmap to display the mutation information for samples.

Usage

```
draw.oncoprint(
  phenotype_info = NULL,
  Missense_column = NULL,
  Missense_label = NULL,
  Amplification_column = NULL,
  Amplification_label = NULL,
  Deletion_column = NULL,
  Deletion_label = NULL,
  Sample_column = NULL,
  main = "",
  pdf_file = NULL,
  ...
)

draw.oncoprint(
  phenotype_info = NULL,
  Missense_column = NULL,
  Missense_label = NULL,
  Amplification_column = NULL,
  Amplification_label = NULL,
  Deletion_column = NULL,
  Deletion_label = NULL,
  Sample_column = NULL,
  main = "",
  pdf_file = NULL,
  ...
)
```

Arguments

phenotype_info	data.frame, phenotype of samples. Users can call Biobase::pData(eset) to create.
Missense_column	character, column names from phenotype_info.
Missense_label,	character, gene label for the Missense_column.
Amplification_column	character, column names from phenotype_info.
Amplification_label,	character, gene label for the Amplification_column.
Deletion_column	character, column names from phenotype_info.
Deletion_label,	character, gene label for the mDeletion_column.
Sample_column	character, column names from phenotype_info. If not NULL, will show sample names in the figure.
main	character, an overall title for the plot. Default is "".
pdf_file	character, the file path to save plot as PDF file. If NULL, no PDF file will be saved. Default is NULL.
...,	for more options, please check ?oncoPrint for more details.

Value

Return a logical value. If TRUE, the plot has been created successfully.

Return a logical value. If TRUE, the plot has been created successfully.

Examples

```
all_sample <- sprintf('Sample%s',1:30)
group1_sample <- sample(all_sample,18) ## demo sample for KRAS missense mutation
group2_sample <- sample(all_sample,12) ## demo sample for MYC amplification
group3_sample <- sample(all_sample,10) ## demo sample for MYC missense mutation
group4_sample <- sample(all_sample,1) ## demo sample for MYC deletion
phenotype_info_demo <-
  data.frame(sample =sprintf('Sample%s',1:30),
             KRAS_MIS=ifelse(all_sample %in% group1_sample,1,0),
             MYC_AMP=ifelse(all_sample %in% group2_sample,1,0),
             MYC_MIS=ifelse(all_sample %in% group3_sample,1,0),
             MYC_DEL=ifelse(all_sample %in% group4_sample,1,0))
draw.oncoprint(phenotype_info=phenotype_info_demo,
               Missense_column=c('KRAS_MIS','MYC_MIS'),Missense_label=c('KRAS','MYC'),
               Amplification_column=c('MYC_AMP'),Amplification_label=c('MYC'),
               Deletion_column=c('MYC_DEL'),Deletion_label=c('MYC'),
               Sample_column='sample',
               main="OncoPrint for the demo dataset")
## Not run:

all_sample <- sprintf('Sample%s',1:30)
group1_sample <- sample(all_sample,18) ## demo sample for KRAS missense mutation
group2_sample <- sample(all_sample,12) ## demo sample for MYC amplification
group3_sample <- sample(all_sample,10) ## demo sample for MYC missense mutation
```

```
group4_sample <- sample(all_sample,1) ## demo sample for MYC deletion
phenotype_info_demo <-
  data.frame(sample =sprintf('Sample%s',1:30),
             KRAS_MIS=ifelse(all_sample %in% group1_sample,1,0),
             MYC_AMP=ifelse(all_sample %in% group2_sample,1,0),
             MYC_MIS=ifelse(all_sample %in% group3_sample,1,0),
             MYC_DEL=ifelse(all_sample %in% group4_sample,1,0))
draw.oncoprint(phenotype_info=phenotype_info_demo,
               Missense_column=c('KRAS_MIS','MYC_MIS'),Missense_label=c('KRAS','MYC'),
               Amplification_column=c('MYC_AMP'),Amplification_label=c('MYC'),
               Deletion_column=c('MYC_DEL'),Deletion_label=c('MYC'),
               Sample_column='sample',
               main="OncoPrint for the demo dataset")
## Not run:
```

`draw.targetNet` *Target Network Structure Plot for One Driver*

Description

`draw.targetNet` draws a network structure to display the target genes of one selected driver. Edges of positively-regulated target genes are orange, edges of negatively-regulated target genes are green. The width of the edges shows the strength of regulation.

`draw.targetNet` draws a network structure to display the target genes of one selected driver. Edges of positively-regulated target genes are orange, edges of negatively-regulated target genes are green. The width of the edges shows the strength of regulation.

Usage

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

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

Arguments

source_label	character, the label of selected one driver.
source_z	numeric, the Z-statistic of the selected driver. The color shade of driver's node in the network is decided by this Z-statistic. If NULL, the driver node will be colored grey. Default is NULL.
edge_score	a named vector of numerics, indicating the correlation between the driver and its target genes. The range of the numeric value is from -1 to 1. Positive value means it is positively-regulated by driver and vice versa. The names of the vector are gene names.
label_cex	numeric, giving the amount by which the text of target gene names should be magnified relative to the default. Default is 0.7.
source_cex	numeric, giving the amount by which the text of driver name should be magnified relative to the default. Default is 1.
pdf_file	character, the file path to save as PDF file. If NULL, no PDF file will be saved. Default is NULL.
arrow_direction	character, users can choose between "in" and "out". Default is "out".
n_layer	integer, number of circle layers to display. Default is 1.
alphabetical_order	logical, if TRUE, the targe gene names will be sorted alphabetically. If FALSE, will be sorted by statistics. Default is FALSE.

Value

Return a logical value. If TRUE, the plot has been created successfully.

Return a logical value. If TRUE, the plot has been created successfully.

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,n_layer=2)
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)
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,n_layer=2)
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 Ntwork Structure Plot for Two Drivers*

Description

`draw.targetNet.TWO` draws a network structure to display the target genes of two selected drivers. Edges of positively-regulated target genes are orange, edges of negatively-regulated target genes are green. The width of the edges shows the strength of regulation. It will also print out the number of shared and unique targe genes for each driver, with P-value and odds ratio.

`draw.targetNet.TWO` draws a network structure to display the target genes of two selected drivers. Edges of positively-regulated target genes are orange, edges of negatively-regulated target genes are green. The width of the edges shows the strength of regulation. It will also print out the number of shared and unique targe genes for each driver, with P-value and odds ratio.

Usage

```
draw.targetNet.TWO(
  source1_label = "",
  source2_label = "",
  source1_z = NULL,
  source2_z = NULL,
  edge_score1 = NULL,
  edge_score2 = NULL,
  arrow_direction1 = "out",
  arrow_direction2 = "out",
```

```
  label_cex = 0.7,
  source_cex = 1,
  pdf_file = NULL,
  total_possible_target = NULL,
  show_test = FALSE,
  n_layer = 1,
  alphabetical_order = FALSE
)

draw.targetNet.TWO(
  source1_label = "",
  source2_label = "",
  source1_z = NULL,
  source2_z = NULL,
  edge_score1 = NULL,
  edge_score2 = NULL,
  arrow_direction1 = "out",
  arrow_direction2 = "out",
  label_cex = 0.7,
  source_cex = 1,
  pdf_file = NULL,
  total_possible_target = NULL,
  show_test = FALSE,
  n_layer = 1,
  alphabetical_order = FALSE
)
```

Arguments

source1_label	character, the label of the first selected driver (to be displayed on the left).
source2_label	character, the label of the second selected driver (to be displayed on the right).
source1_z	numeric, the Z-statistic of the first driver. The color shade of driver's node in the network is decided by this Z-statistic. If NULL, the driver will be colored grey. Default is NULL.
source2_z	numeric, the Z-statistic of the second driver.The color shade of driver's node in the network is decided by this Z-statistic. If NULL, the driver will be colored grey. Default is NULL.
edge_score1	a named vector of numerics, indicating the correlation between the first driver and its target genes. The range of the numeric value is from -1 to 1. Positive value means it is positively-regulated by driver and vice versa. The names of the vector are gene names.
edge_score2	a named vector of numerics, indicating the correlation between the seconde driver and its target genes. The range of the numeric value is from -1 to 1. Positive value means it is positively-regulated by driver and vice versa. The names of the vector are gene names.
arrow_direction1	character, the arrow direction for first driver. Users can choose between "in" and "out". Default is "out".
arrow_direction2	character, the arrow direction for second driver. Users can choose between "in" and "out". Default is "out".

label_cex numeric, giving the amount by which the text of target gene names should be magnified relative to the default. Default is 0.7.

source_cex numeric, giving the amount by which the text of driver name should be magnified relative to the default. Default is 1.

pdf_file character, the file path to save as PDF file. If NULL, no PDF file will be saved. Default is NULL.

total_possible_target
numeric or a vector of characters. If input is numeric, it is the total number of possible target genes. If input is a vector of characters, it is 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, if TRUE, the test result will be printed and returned. Default is FALSE.

n_layer integer, number of circle layers to display. Default is 1.

alphabetical_order
logical, if TRUE, the targe gene names will be sorted alphabetically. If FALSE, will be sorted by statistics. Default is FALSE.

Value

If show_test==FALSE, will return a logical value indicating whether the plot has been successfully generated, otherwise will return the statistics of testing when total_possible_target is not NULL.

If show_test==FALSE, will return a logical value indicating whether the plot has been successfully generated, otherwise will return the statistics of testing when total_possible_target is not NULL.

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)
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,n_layer=2)

## 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=100)
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=100)
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)
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)
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,n_layer=2)

## 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=100)
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=100)
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.volcanoPlot` *Draw Volcano Plot for Top DE (differentiated expressed) Genes or DA (differentiated activity) Drivers*

Description

`draw.volcanoPlot` draws the volcano plot to identify and visualize DE genes or DA drivers with fold change threshold and significant P-value from the input dataset. The function will return a data.frame of these highlighted genes/drivers.

`draw.volcanoPlot` draws the volcano plot to identify and visualize DE genes or DA drivers with fold change threshold and significant P-value from the input dataset. The function will return a data.frame of these highlighted genes/drivers.

Usage

```
draw.volcanoPlot(
  dat = NULL,
  label_col = NULL,
  logFC_col = NULL,
  Pv_col = NULL,
  logFC_thre = 0.1,
  Pv_thre = 0.01,
  show_plot = TRUE,
  xlab = "log2 Fold Change",
  ylab = "P-value",
  show_label = FALSE,
  label_cex = 0.5,
  legend_cex = 0.8,
  label_type = "distribute",
  main = "",
  pdf_file = NULL
)

draw.volcanoPlot(
  dat = NULL,
  label_col = NULL,
  logFC_col = NULL,
  Pv_col = NULL,
  logFC_thre = 0.1,
  Pv_thre = 0.01,
  show_plot = TRUE,
  xlab = "log2 Fold Change",
  ylab = "P-value",
  show_label = FALSE,
  label_cex = 0.5,
  legend_cex = 0.8,
  label_type = "distribute",
  main = "",
  pdf_file = NULL
)
```

Arguments

dat	data.frame, the master table created by function generate.masterTable. Or a table with columns of the following parameters.
label_col	character, the name of the column in dat contains gene/driver names.
logFC_col	character, the name of the column in dat contains logFC values.
Pv_col	character, the name of the column in dat contains P-values.
logFC_thre	numeric, the threshold of logFC. Genes or drivers with absolute logFC value higher than the threshold will be kept. Default is 0.1.
Pv_thre	numeric, the threshold of P-values. Genes or drivers with P-values lower than the threshold will be kept. Default is 0.01.
show_plot	logical, if TRUE, the plot will be shown in the plot pane. Default is TRUE.
xlab	character, a title for the X axis.
ylab	character, a title for the Y axis.
show_label	logical, if TRUE, labels of selected genes or drivers will be displayed in the plot. Default is FALSE.
label_cex	numeric, giving the amount by which the text of genes/drivers label should be magnified relative to the default. Default is 0.5.
legend_cex	numeric, giving the amount by which the text of legend should be magnified relative to the default. Default is 0.7.
label_type	character, users can choose between "origin" and "distribute". If "origin", all the labels will be displayed without location modification. If "distribute", location of labels will be rearranged to avoid overlap. Default is "distribute".
main	character, an overall title for the plot.
pdf_file	character, the path to save the plot as PDF file. If NULL, no PDF file will be created. Default is NULL.

Details

Top genes or drivers will be colored (blue for down-regulated and red for up-regulated) and labeled with their names. This function requires the input of master table and two thresholds of logFC and P-value.

Top genes or drivers will be colored (blue for down-regulated and red for up-regulated) and labeled with their names. This function requires the input of master table and two thresholds of logFC and P-value.

Value

Return a data.frame of selected significant genes or drivers, with columns contain label_col, logFC_col and Pv_col.

Return a data.frame of selected significant genes or drivers, with columns contain label_col, logFC_col and 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)
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)
```

`funcEnrich.Fisher` *Gene Set Enrichment Analysis by Fisher's Exact Test*

Description

`funcEnrich.Fisher` performs gene set enrichment analysis to the input gene list, by using the Fisher's Exact Test. Background gene list is accepeted.

`funcEnrich.Fisher` performs gene set enrichment analysis to the input gene list, by using the Fisher's Exact Test. Background gene list is accepeted.

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
)

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, a vector of gene symbols. If gene symbols are not available, users can call get_IDtransfer and get_name_transfertab for ID conversion.
bg_list	a vector of characters, a vector of background gene symbols. If NULL, genes in gs2gene will be used as background. Default is NULL.
use_gs	a vector of characters, the names of gene sets. If gs2gene is NULL, all_gs2gene will be used. The use_gs must be the subset of names(all_gs2gene). If "all", all the gene sets in gs2gene will be used. If user input his own gs2gene list, use_gs will be set to "all" as default. Default is c("H", "CP:BIOCARTA", "CP:REACTOME", "CP:KEGG").
gs2gene	list, a list contains elements of gene sets. The name of the element is gene set, each element contains a vector of genes in that gene set. If NULL, will use all_gs2gene, which is created by function gs.preload. Default is NULL.
min_gs_size	numeric, the minimum size of gene set to analysis. Default is 5.
max_gs_size	numeric, the maximum size of gene set to analysis. Default is 500.
Pv_adj	character, method to adjust P-value. Default is "fdr". For details, please check p.adjust.methods.
Pv_thre	numeric, threshold for the adjusted P-values. Default is 0.1.

Value

Return a data.frame, contains gene sets with significant enrichment statistics. Column details are as follows,

#Name	Name of the enriched gene set
Total_item	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 of 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 from the 2*2 matrix used for Fisher's Exact Test
Intersected_items	A vector of the intersected genes, collapsed by ';'. Number is equal to Num_list_item

Return a data.frame, contains gene sets with significant enrichment statistics. Column details are as follows,

#Name	Name of the enriched gene set
Total_item	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 of 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 from the 2*2 matrix used for Fisher's Exact Test

Intersected_items
A vector of the intersected genes, collapsed by ';'. 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:

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:
```

---

funcEnrich.GSEA *Gene Set Enrichment Analysis by GSEA*

Description

funcEnrich.GSEA performs gene set enrichment analysis to the input gene list, by using the GSEA Test.

funcEnrich.GSEA performs gene set enrichment analysis to the input gene list, by using the GSEA Test.

Usage

```
funcEnrich.GSEA(
  rank_profile = NULL,
  use_gs = NULL,
  gs2gene = NULL,
  min_gs_size = 5,
  max_gs_size = 500,
  Pv_adj = "fdr",
  Pv_thre = 0.1,
  test_strategy = "GSEA",
  nperm = 1000,
  use_seed = 999
)

funcEnrich.GSEA(
  rank_profile = NULL,
  use_gs = NULL,
  gs2gene = NULL,
  min_gs_size = 5,
  max_gs_size = 500,
  Pv_adj = "fdr",
  Pv_thre = 0.1,
  test_strategy = "GSEA",
  nperm = 1000,
  use_seed = 999
)
```

Arguments

rank_profile	a named vector of numerics, the differential values (DE or DA) calculated from a sample comparison (e.g. "G4 vs. Others"). Names of the vector must be gene names. For the DA, user could use 'processDriverProfile()' to convert the DA profile into gene-name based profile. The differential values can be "logFC" or "t-statistics".
use_gs	a vector of characters, the names of gene sets. If gs2gene is NULL, all_gs2gene will be used. The use_gs must be the subset of names(all_gs2gene). If "all", all the gene sets in gs2gene will be used. If user input his own gs2gene list, use_gs will be set to "all" as default. Default is c("H", "CP:BIOCARTA", "CP:REACTOME", "CP:KEGG").
gs2gene	list, a list contains elements of gene sets. The name of the element is gene set, each element contains a vector of genes in that gene set. If NULL, will use all_gs2gene, which is created by function gs.preload. Default is NULL.
min_gs_size	numeric, the minimum size of gene set to analysis. Default is 5.
max_gs_size	numeric, the maximum size of gene set to analysis. Default is 500.

Pv_adj	character, method to adjust P-value. Default is "fdr". For details, please check p.adjust.methods.
Pv_thre	numeric, threshold for the adjusted P-values. Default is 0.1.
test_strategy	choose from "KS" and "GSEA". Default is "GSEA". If "KS", will perform a Kolmogorov-Smirnov test to get the significance value.
nperm	numeric, number of random permutations. Default is 1000. This function only do gene-label based permutation reshuffling.
use_seed	integer, the random seed. Default is 999.

Value

Return a data.frame, contains gene sets with significant enrichment statistics. Column details are as follows (test_strategy=GSEA),

#Name	Name of the enriched gene set
Total_item	Size in the profile
Num_item	Number of genes in the gene set (filtered by the profile list)
Ori_P	Original P-value from GSEA Test
Adj_P	Adjusted P-value
ES	Enrichment Score
NES	normalized Enrichment Score

Return a data.frame, contains gene sets with significant enrichment statistics. Column details are as follows (test_strategy=GSEA),

#Name	Name of the enriched gene set
Total_item	Size in the profile
Num_item	Number of genes in the gene set (filtered by the profile list)
Ori_P	Original P-value from GSEA Test
Adj_P	Adjusted P-value
ES	Enrichment Score
NES	normalized Enrichment Score

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
## get significant gene set by driver's DA profile
DA_profile <- processDriverProfile(Driver_name=ms_tab$gene_label,
                                   Driver_profile=ms_tab$logFC.G4.Vs.others_DA,
                                   choose_strategy='absmax',
                                   return_type ='gene_statistics')
res1 <- funcEnrich.GSEA(rank_profile=DA_profile,
                        use_gs=c('H'),
                        Pv_thre=0.1,Pv_adj = 'none')
## 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
## get significant gene set by driver's DA profile
DA_profile <- processDriverProfile(Driver_name=ms_tab$gene_label,
                                   Driver_profile=ms_tab$logFC.G4.Vs.others_DA,
                                   choose_strategy='absmax',
                                   return_type ='gene_statistics')
res1 <- funcEnrich.GSEA(rank_profile=DA_profile,
                        use_gs=c('H'),
                        Pv_thre=0.1,Pv_adj = 'none')
## Not run:
```

`generate.eset` *Generate ExpressionSet Object*

Description

`generate.eset` generates ExpressionSet class object to contain and describe the high-throughput assays. Users need to define its slots, which are expression matrix (required), phenotype information and feature information (optional). It is very useful when only expression matrix is available.

`generate.eset` generates ExpressionSet class object to contain and describe the high-throughput assays. Users need to define its slots, which are expression matrix (required), phenotype information and feature information (optional). It is very useful when only expression matrix is available.

Usage

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

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

Arguments

`exp_mat`	matrix, the expression data matrix. Each row represents a gene/transcript/probe, each column represents a sample.
`phenotype_info`	data.frame, the phenotype information for all the samples in `exp_mat`. In the phenotype data frame, each row represents a sample, each column represents a phenotype feature. The row names must match the column names of `exp_mat`. If NULL, it will generate a single-column data frame. Default is NULL.

feature_info data.frame, the feature information for all the genes/transcripts/probes in exp_mat. In the feature data frame, each row represents a gene/transcript/probe and each column represents an annotation of the feature. The row names must match the row names of exp_mat. If NULL, it will generate a single-column data frame. Default is NULL.

annotation_info character, the annotation set by users for easier reference. Default is "".

Value

Return an ExressionSet object.

Return 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)
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 the Master Table for Drivers*

Description

generate.masterTable generates a master table to show the mega information of all tested drivers.

generate.masterTable generates a master table to show the mega information of all tested drivers.

Usage

```
generate.masterTable(
  use_comp = NULL,
  DE = NULL,
  DA = NULL,
  target_list = 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"
)

generate.masterTable(
  use_comp = NULL,
  DE = NULL,
  DA = NULL,
```

```
  target_list = 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 name of multiple comparisons. It will be used to name the columns of master table.
DE	list, a list of DE comparisons, each comparison is a data.frame. The element name in the list must contain the name in use_comp.
DA	list, a list of DA comparisons, each comparison is a data.frame. The element name in the list must contain the name in use_comp.
target_list	list, a driver-to-target list. The names of the list elements are drivers. Each element is a data frame, usually contains three columns. "target", target gene names; "MI", mutual information; "spearman", spearman correlation coefficient. It is highly suggested to follow the NetBID2 pipeline, and the TF_network could be generated by get_net2target_list and get.SJAracne.network.
main_id_type	character, the type of driver's ID. It comes from the attribute name in biomaRt package. Such as "ensembl_gene_id", "ensembl_gene_id_version", "ensembl_transcript_id", "ensembl_transcript_id_version" or "refseq_mrna". For details, user can call biomaRt::listAttributes() to display all available attributes in the selected dataset.
transfer_tab	data.frame, the data frame for ID conversion. This can be obtained by calling get_IDtransfer. If NULL and main_id_type is not in the column names of tf_sigs, it will use the conversion table within the function. Default is NULL.
tf_sigs	list, contains all the detailed information of TF and Sig. Users can call db.preload for access.
z_col	character, name of the column in DE and DA contains the Z statistics. Default is "Z-statistics".
display_col	character, name of the column in DE and DA need to be kept in the master table. Default is c("logFC","P.Value").
column_order_stratey	character, users can choose between "type" and "comp". Default is "type". If set as type, the columns will be ordered by column type; If set as comp, the columns will be ordered by comparison.

Details

The master table gathers TF (transcription factor) information, Sig (signaling factor) information, all the DE (differential expression analysis) and DA (differential activity analysis) from multiple comparisons. It also shows each driver's target gene size and other additional information (e.g. gene biotype, chromosome name, position etc.).

The master table gathers TF (transcription factor) information, Sig (signaling factor) information, all the DE (differential expression analysis) and DA (differential activity analysis) from multiple comparisons. It also shows each driver's target gene size and other additional information (e.g. gene biotype, chromosome name, position etc.).

Value

Return a data frame contains the mega information of all tested drivers. The column "originalID" and "originalID_label" is the same ID as from the original dataset.

Return a data frame contains the mega information of all tested drivers. The column "originalID" and "originalID_label" is the same ID as from 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=Biobase::exprs(analysis.par$cal.eset),es.method='weightedmean')
analysis.par$ac.merge.eset  <- generate.eset(exp_mat=ac_mat,
                                  phenotype_info=Biobase::pData(analysis.par$cal.eset))
phe_info <- Biobase::pData(analysis.par$cal.eset)
all_subgroup <- base::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,
                                    target_list=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')
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=Biobase::exprs(analysis.par$cal.eset),es.method='weightedmean')
analysis.par$ac.merge.eset  <- generate.eset(exp_mat=ac_mat,
                                  phenotype_info=Biobase::pData(analysis.par$cal.eset))
phe_info <- Biobase::pData(analysis.par$cal.eset)
all_subgroup <- base::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$DA,
                                  target_list=analysis.par$merge.network$target_list,
                                        tf_sigs=tf_sigs,
                                        z_col='Z-statistics',
                                        display_col=c('logFC','P.Value'),
                                        main_id_type='external_gene_name')
```

`get.class.color` *Create Color Codes for a Vector of Characters*

Description

`get.class.color` creates a vector of color codes for the input character vector. This is a helper function to assign nice looking colors for better visualization.

`get.class.color` creates a vector of color codes for the input character vector. This is a helper function to assign nice looking colors for better visualization.

Usage

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

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

Arguments

`x`	a vector of characters, names or labels.
`use_color`	a vector of color codes, colors to be assigned to each member of `x`. Default is brewer.pal(9, 'Set1').
`pre_define`	a vector of characters, pre-defined color codes for a certain input (e.g. c("blue", "red") with names c("A", "B")). Default is NULL.

Value

Return a vector of color codes, with input character vector as names.

Return a vector of color codes, with input character vector as names.

Examples

```
get.class.color(c('ClassA','ClassB','ClassC','ClassA','ClassC','ClassC'))
get.class.color(c('ClassA','ClassB','ClassC','SHH','WNT','Group3','Group4'))
get.class.color(c('ClassA','ClassB','ClassC','SHH','WNT','Group3','Group4'),
                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)

## Not run:

get.class.color(c('ClassA','ClassB','ClassC','ClassA','ClassC','ClassC'))
get.class.color(c('ClassA','ClassB','ClassC','SHH','WNT','Group3','Group4'))
get.class.color(c('ClassA','ClassB','ClassC','SHH','WNT','Group3','Group4'),
                 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)

## Not run:
```

`get.SJAracne.network` *Read SJARACNe Network Result and Return it as List Object*

Description

`get.SJAracne.network` reads SJARACNe network construction result and returns a list object with network data frame, driver-to-target list and igraph object wrapped inside.

`get.SJAracne.network` reads SJARACNe network construction result and returns a list object with network data frame, driver-to-target list and igraph object wrapped inside.

Usage

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

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

Arguments

`network_file`	character, the path for storing network file. For the output of SJAracne, the name of the network file will be "consensus_network_ncol_.txt" under the output directory.

Details

In the demo, "consensus_network_ncol_.txt" file will be read and convert into a list object. This list contains three elements, `network_data`, `target_list` and `igraph_obj`. `network_dat` is a data.frame, contains all the information of the network SJARACNe constructed. `target_list` is a driver-to-target list object. Please check details in `get_net2target_list`. `igraph_obj` is an igraph object used to save this directed and weighted network. Each edge of the network has two

attributes, weight and sign. weight is the "MI (mutual information)" value and sign is the sign of the spearman correlation coefficient (1, positive regulation; -1, negative regulation).

In the demo, "consensus_network_ncol_.txt" file will be read and convert into a list object. This list contains three elements, network_data, target_list and igraph_obj. network_dat is a data.frame, contains all the information of the network SJARACNe constructed. target_list is a driver-to-target list object. Please check details in get_net2target_list. igraph_obj is an igraph object used to save this directed and weighted network. Each edge of the network has two attributes, weight and sign. weight is the "MI (mutual information)" value and sign is the sign of the spearman correlation coefficient (1, positive regulation; -1, negative regulation).

Value

Return a list containing three elements, network_dat, target_list and igraph_obj.

Return a list containing three elements, 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,
                                            project_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:

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,
                                            project_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*

Description

get.TF_SIG.list is a function converts gene ID into the corresponding TF/SIG list, with selected gene/transcript type.

get.TF_SIG.list is a function converts gene ID into the corresponding TF/SIG list, with selected gene/transcript type.

Usage

```
get.TF_SIG.list(
  use_genes = NULL,
  use_gene_type = "external_gene_name",
  ignore_version = FALSE,
  dataset = NULL
)

get.TF_SIG.list(
  use_genes = NULL,
  use_gene_type = "external_gene_name",
  ignore_version = FALSE,
  dataset = NULL
)
```

Arguments

use_genes	a vector of characters, genes will be used in the network construction. If NULL, no filter will be performed to the TF/SIG list. Default is NULL.
use_gene_type	character, the attribute name inherited from the biomaRt package. Some options are, "ensembl_gene_id", "ensembl_gene_id_version", "ensembl_transcript_id", "ensembl_transcript_id_version" and "refseq_mrna". All options can be accessed by calling biomaRt::useMart (e.g. mart <- biomaRt::useMart('ensembl',db_info[1]); biomaRt::listAttributes(mart)$name). The type must match the gene type from the input use_genes. Default is "external_gene_name".
ignore_version	logical, if TRUE, the version "ensembl_gene_id_version" or "ensembl_transcript_id_version" will be ignored. Default is FALSE.
dataset	character, the dataset used for ID conversion (e.g. "hsapiens_gene_ensembl"). If NULL, use db_info[1] from db.preload. Default is NULL.

Value

Return a list containing two elements. tf is the TF list, sig is the SIG list.

Return a list containing two elements. tf is the TF list, sig is 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:


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` *Differential Expression Analysis and Differential Activity Analysis Between 2 Sample Groups Using Bayesian Inference*

Description

`getDE.BID.2G` is a function performs differential gene expression analysis and differential driver activity analysis between control group (parameter G0) and experimental group (parameter G1).

`getDE.BID.2G` is a function performs differential gene expression analysis and differential driver activity analysis between control group (parameter G0) and experimental group (parameter G1).

Usage

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

getDE.BID.2G(
  eset,
  output_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, contains gene expression data or driver activity data.
output_id_column	character, the column names of Biobase::fData(eset). This option is useful when the eset expression matrix is at transcript-level, and user is expecting to see the gene-level comparison statistics. If NULL, rownames of the Biobase::fData(eset) will be used. Default is NULL.
G1	a vector of characters, the sample names of experimental group.
G0	a vecotr of characters, the sample names of control group.
G1_name	character, the name of experimental group (e.g. "Male"). Default is "G1".
G0_name	character, the name of control group (e.g. "Female"). Default is "G0".
method	character, users can choose between "MLE" and "Bayesian". "MLE", the maximum likelihood estimation, will call generalized linear model(glm/glmer) to perform data regression. "Bayesian", will call Bayesian generalized linear model (bayesglm) or multivariate generalized linear mixed model (MCMCglmm) to perform data regression. Default is "Bayesian".
family	character or family function or the result of a call to a family function. This parameter is used to define the model's error distribution. See ?family for details. Currently, options are gaussian, poisson, binomial(for two-group sample classes)/category(for multi-group sample classes)/ordinal(for multi-group sample classes with class_ordered=TRUE). If set with gaussian or poission, the response variable in the regression model will be the expression level, and the independent variable will be the sample's phenotype. If set with binomial, the response variable in the regression model will be the sample phenotype, and the independent variable will be the expression level. For binomial, category and ordinal input, the family will be automatically reset, based on the sample's class level and the setting of class_ordered. Default is gaussian.
pooling	character, users can choose from "full","no" and "partial". "full", use probes as independent observations. "no", use probes as independent variables in the regression model. "partial", use probes as random effect in the regression model. Default is "full".
logTransformed	logical, if TRUE, log tranformation of the expression value will be performed.
verbose	logical, if TRUE, sample names of both groups will be printed. Default is TRUE.

Value

Return a data frame. Rows are genes/drivers, columns are "ID", "logFC", "AveExpr", "t", "P.Value", "adj.P.Val", "Z-statistics", "Ave.G1" and "Ave.G0". Names of the columns may vary from different group names. Sorted by P.Value.

Return a data frame. Rows are genes/drivers, columns are "ID", "logFC", "AveExpr", "t", "P.Value", "adj.P.Val", "Z-statistics", "Ave.G1" and "Ave.G0". Names of the columns may vary from different group names. Sorted by P.Value.

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,output_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,output_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,output_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$out.dir.DATA <- system.file('demo1','driver/DATA/',package = "NetBID2")
NetBID.loadRData(analysis.par=analysis.par,step='ms-tab')
phe_info <- Biobase::pData(analysis.par$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)
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,output_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,output_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,output_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$out.dir.DATA <- system.file('demo1','driver/DATA/',package = "NetBID2")
NetBID.loadRData(analysis.par=analysis.par,step='ms-tab')
phe_info <- Biobase::pData(analysis.par$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` *Differential Expression Analysis and Differential Activity Analysis Between 2 Sample Groups Using Limma*

Description

`getDE.limma.2G` is a function performs differential gene expression analysis and differential driver activity analysis between control group (parameter G0) and experimental group (parameter G1), using limma related functions.

`getDE.limma.2G` is a function performs differential gene expression analysis and differential driver activity analysis between control group (parameter G0) and experimental group (parameter G1), using limma related functions.

Usage

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

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, contains gene expression data or driver activity data.
G1	a vector of characters, the sample names of experimental group.
G0	a vecotr of characters, the sample names of control group.
G1_name	character, the name of experimental group (e.g. "Male"). Default is "G1".
G0_name	character, the name of control group (e.g. "Female"). Default is "G0".
verbose	logical, if TRUE, sample names of both groups will be printed. Default is TRUE.
random_effect	a vector of characters, vector or factor specifying a blocking variable. Default is NULL, no random effect will be considered.

Value

Return a data frame. Rows are genes/drivers, columns are "ID", "logFC", "AveExpr", "t", "P.Value", "adj.P.Val", "B", "Z-statistics", "Ave.G1" and "Ave.G0". Names of the columns may vary from different group names. Sorted by P-values.

Return a data frame. Rows are genes/drivers, columns are "ID", "logFC", "AveExpr", "t", "P.Value", "adj.P.Val", "B", "Z-statistics", "Ave.G1" and "Ave.G0". Names of the columns may vary from different group names. Sorted by P-values.

Examples

```
analysis.par <- list()
analysis.par$out.dir.DATA <- system.file('demo1','driver/DATA/',package = "NetBID2")
NetBID.loadRDData(analysis.par=analysis.par,step='ms-tab')
phe_info <- Biobase::pData(analysis.par$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')
analysis.par <- list()
analysis.par$out.dir.DATA <- system.file('demo1','driver/DATA/',package = "NetBID2")
NetBID.loadRDData(analysis.par=analysis.par,step='ms-tab')
phe_info <- Biobase::pData(analysis.par$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 to Measure Similarity Between Observed Classification and Predicted Classification.*

Description

`get_clustComp` calculates a score to measure the similarity between two classifications.

`get_clustComp` calculates a score to measure the similarity between two classifications.

Usage

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


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


Arguments

`pred_label`	a vector of characters, the predicted classification labels.
`obs_label`	a vector of characters, the observed classification labels.
`strategy`	character, the method applied to calculate the score. Users can choose "ARI (adjusted rand index)", "NMI (normalized mutual information)" or "Jaccard". Default is "ARI".

Value

Return a score for the measurement of similarity.

Return a score for the measurement of similarity.

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)
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 *Creates Data Frame for ID Conversion*

Description

get_IDtransfer creates a data frame for ID conversion using biomaRt. For example, to convert Ensembl ID into gene symbol.

get_IDtransfer creates a data frame for ID conversion using biomaRt. For example, to convert Ensembl ID into gene symbol.

Usage

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

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

Arguments

from_type	character, the attribute name match the current ID type (the type of use_genes). Such as "ensembl_gene_id", "ensembl_gene_id_version", "ensembl_transcript_id", "ensembl_transcript_id_version" or "refseq_mrna". The "attribute" is inherited from the biomaRt package. For details, user can call biomaRt::listAttributes() to display all available attributes in the selected dataset.
to_type	character, the attribute name to convert into.
add_type	character, the additional attribute name to add into the conversion data frame.
use_genes	a vector of characters, the genes for ID conversion. If NULL, all genes will be selected.
dataset	character, name of the dataset used for ID conversion. For example, "hsapiens_gene_ensembl". If NULL, db_info[1] will be used. db_info requires the calling of db.preload in the previous steps. Default is NULL.
ignore_version	logical, if it is set to TRUE and from_type is "ensembl_gene_id_version" or "ensembl_transcript_id_version", the version of the original ID will be ignored in ID mapping. Default is FALSE.

Value

Return a data frame for ID conversion.

Return a data frame for ID conversion.

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:

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

Create Data Frame for ID Conversion With Biotype Information

Description

get_IDtransfer2symbol2type creates a data frame to convert original ID into gene symbol and gene biotype (gene level), or into transcript symbol and transcript biotype (transcript level).

get_IDtransfer2symbol2type creates a data frame to convert original ID into gene symbol and gene biotype (gene level), or into transcript symbol and transcript biotype (transcript level).

Usage

```
get_IDtransfer2symbol2type(
  from_type = NULL,
  use_genes = NULL,
  dataset = NULL,
```

```
  use_level = "gene",
  ignore_version = FALSE
)

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

Arguments

from_type	character, the attribute name matches the current ID type (the type of use_genes). Such as "ensembl_gene_id", "ensembl_gene_id_version", "ensembl_transcript_id", "ensembl_transcript_id_version" or "refseq_mrna". The "attribute" is inherited from the biomaRt package. For details, user can call `biomaRt::listAttributes()` function to display all available attributes in the selected dataset.
use_genes	a vector of characters, the genes for ID conversion. If NULL, all genes will be selected.
dataset	character, name of the dataset used for ID conversion. For example, "hsapiens_gene_ensembl". If NULL, `db_info[1]` will be used. `db_info` requires the calling of `db.preload` in the previous steps. Default is NULL.
use_level	character, users can chose between "transcript" and "gene". Default is "gene".
ignore_version	logical, if it is set to TRUE and `from_type` is "ensembl_gene_id_version" or "ensembl_transcript_id_version", the version of the original ID will be ignored in ID mapping.

Value

Return a data frame for ID conversion, from ID to gene symbol and gene biotype.

Return a data frame for ID conversion, from ID to gene symbol and gene 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:

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` *Create Data Frame for ID Conversion Between Species*

Description

`get_IDtransfer_betweenSpecies` creates a data frame to convert ID between species.

`get_IDtransfer_betweenSpecies` creates a data frame to convert ID between species.

Usage

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

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

Arguments

`from_spe`	character, name of the original species (e.g. "human", "mouse", "rat") that `use_genes` belongs to. Default is "human".
`to_spe`	character, name of the target species (e.g. "human", "mouse", "rat"). Default is "mouse".
`from_type`	character, the attribute name match the current ID type (the type of `use_genes`). Such as "ensembl_gene_id", "ensembl_gene_id_version", "ensembl_transcript_id", "ensembl_transcript_id_version" or "refseq_mrna". The "attribute" is inherited from the `biomaRt` package. For details, user can call `biomaRt::listAttributes()` function to display all available attributes in the selected dataset.
`to_type`	character, the attribute name match the target ID type.

use_genes a vector of characters, the genes for ID conversion. Must be the genes with ID type of from_type, and from species from_spe. If NULL, all the possible genes will be shown in the conversion table. Default is NULL.

Value

Return a data frame for ID conversion, from one species to another.

Return a data frame for ID conversion, from one species to another.

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)
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 phenotype groups from pData slot of the ExpressionSet object.*

Description

get_int_group is a function to extract interested phenotype groups from the ExpressionSet object with 'cluster-meaningful' sample features.

get_int_group is a function to extract interested phenotype groups from the ExpressionSet object with 'cluster-meaningful' sample features.

Usage

```
get_int_group(eset)

get_int_group(eset)
```

Arguments

eset	an ExpressionSet object.

Value

Return a vector of phenotype groups which could be used for sample cluster analysis.

Return a vector of phenotype groups which could be used for sample cluster analysis.

Examples

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

get_name_transfertab *Convert Original Gene ID into Target Gene ID*

Description

get_name_transfertab converts the original gene IDs into target gene IDs, with conversion table provided.

get_name_transfertab converts the original gene IDs into target gene IDs, with conversion table provided.

Usage

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

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

Arguments

use_genes	a vector of characters, the genes for ID conversion.
transfer_tab	data.frame, the conversion table. Users can create it by calling get_IDtransfer.
from_type	character, the attribute name match the current ID type (the type of use_genes). Such as "ensembl_gene_id", "ensembl_gene_id_version", "ensembl_transcript_id", "ensembl_transcript_id_version" or "refseq_mrna". The "attribute" is inherited from the biomaRt package. For details, user can call biomaRt::listAttributes() to see all available attributes in the selected dataset. If NULL, will use the first column of transfer_tab.
to_type	character, the attribute name to convert into. If NULL, will use the second column of transfer_tab.
ignore_version	logical, if TRUE and from_type is "ensembl_gene_id_version" or "ensembl_transcript_id_version", the version will be ignored. Default is FALSE.
ignore_order	logical, whether need to ignore the output order to match the input list of use_genes. Default is FALSE.

Value

Return a vector of converted gene IDs.

Return a vector of converted gene 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:

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` *Convert Pairwise Network Data Frame to Driver-to-Target List*

Description

`get_net2target_list` is a helper function in the `get.SJAracne.network`. But if users have their own pairwise gene network files, they can convert it to driver-to-target list object.

`get_net2target_list` is a helper function in the `get.SJAracne.network`. But if users have their own pairwise gene network files, they can convert it to driver-to-target list object.

Usage

```
get_net2target_list(net_dat = NULL)

get_net2target_list(net_dat = NULL)
```

Arguments

`net_dat`	data.frame, must contain two columns with column names "source" (driver) and "target" (target genes). "MI" (mutual information) and "spearman" (spearman correlation coefficient) columns are optional, but strongly suggested to use. If "MI" and "spearman" columns are missing, errors may occur in some following steps (e.g. es.method='weightedmean' in `cal.Activity`).

Value

Return a list. The names of the list elements are drivers. Each element is a data frame, contains three columns. "target", target gene names; "MI", mutual information; "spearman", spearman correlation coefficient.

Return a list. The names of the list elements are drivers. Each element is a data frame, contains three columns. "target", target gene names; "MI", mutual information; "spearman", spearman correlation coefficient.

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)
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 *Create a vector of each sample's selected phenotye descriptive information.*

Description

get_obs_label creates a vector of each sample's selected phenotype descriptive information. This is a helper function for data visualization.

get_obs_label creates a vector of each sample's selected phenotype descriptive information. This is a helper function for data visualization.

Usage

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

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

Arguments

phe_info	data.frame, the phenotype data of the samples. It is a data frame that can store any number of descriptive columns (covariates) for each sample row. To get the phenotype data, using the accessor function pData.
use_col	a vector of numerics or characters. Users can select the interested descriptive column(s) by calling index or name of the column(s).
collapse	character, an optional character string to separate the results when the length of use_col is more than 1. Not NA_character. Default is "|".

Value

Return a vector of selected phenotype descriptive information (covariates) for each sample. Vector name is the sample name.

Return a vector of selected phenotype descriptive information (covariates) for each sample. Vector name is the sample name.

Examples

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

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

---

gs.preload *Load MSigDB Database into R Workspace*

Description

gs.preload downloads data from MSigDB and stores it into two variables in R workspace, all_gs2gene and all_gs2gene_info. all_gs2gene is a list object with elements of gene sets collections. all_gs2gene_info is a data.frame contains the description of each gene sets.

gs.preload downloads data from MSigDB and stores it into two variables in R workspace, all_gs2gene and all_gs2gene_info. all_gs2gene is a list object with elements of gene sets collections. all_gs2gene_info is a data.frame contains the description of each gene sets.

Usage

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

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

Arguments

use_spe	character, name of interested species (e.g. "Homo sapiens", "Mus musculus"). Users can call msigdbr_show_species() to access the full list of available species names. Default is "Homo sapiens".

update	logical, if TRUE, the previous loaded RData will be updated. Default is FALSE.
main.dir	character, the main file path of user's NetBID2 project. If NULL, will be set to system.file(package = "NetBID2"). Default is NULL.
db.dir	character, the file path to save the RData. Default is db directory under the main.dir, if one has a 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"). It will call msigdbr to download data from MSigDB and save it as RData under the db/ directory with species name.

This is a pre-processing function for NetBID2 advanced analysis. User only need to input the species name (e.g. "Homo sapiens", "Mus musculus"). It will call msigdbr to download data from MSigDB and save it as RData under the db/ directory with species name.

Value

Reture a logical value. If TRUE, MsigDB database is loaded successfully, with all_gs2gene and all_gs2gene_info created in the workspace.

Reture a logical value. If TRUE, MsigDB database is loaded successfully, with all_gs2gene and all_gs2gene_info created in the workspace.

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)
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 to extract genes from expression matrix*

Description

IQR.filter is a function to extract genes from the expression matrix by setting threshold to their IQR value. IQR (interquartile range) is a measure of statistical dispersion. It is calculated for each gene across all the samples. By setting threshold value, genes with certain statistical dispersion across samples will be filtered out. This step is mainly used to perform sample cluster and to prepare the input for SJAracne.

IQR.filter is a function to extract genes from the expression matrix by setting threshold to their IQR value. IQR (interquartile range) is a measure of statistical dispersion. It is calculated for each gene across all the samples. By setting threshold value, genes with certain statistical dispersion across samples will be filtered out. This step is mainly used to perform sample cluster and to prepare the input for SJAracne.

Usage

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

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. Each row represents a gene/transcript/probe, each column represents a sample.
use_genes	a vector of characters, the gene list needed to be filtered. Default is the row names of exp_mat.
thre	numeric, the threshold for IQR of the genes in use_genes. 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 the input of possible drivers used in SJAracne. Default is NULL.
loose_thre	numeric, the threshold for IQR of the genes in loose_gene. Default is 0.1.

Value

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

Return 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))
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 *Download Gene Expression Series From GEO Database with Platform Specified*

Description

load.exp.GEO downloads user assigned Gene Expression Series (GSE file) along with its Platform from GEO dataset. It returns an ExpressionSet class object and saves it as RData. If the GSE RData already exists, it will be loaded directly. It also allows users to update the Gene Expression Series RData saved before.

load.exp.GEO downloads user assigned Gene Expression Series (GSE file) along with its Platform from GEO dataset. It returns an ExpressionSet class object and saves it as RData. If the GSE RData already exists, it will be loaded directly. It also allows users to update the Gene Expression Series RData saved before.

Usage

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

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

Arguments

out.dir	character, the file path used to save the GSE RData. If the data already exsits, it will be loaded from this path.
GSE	character, the GEO Series Accession ID.
GPL	character, the GEO Platform Accession ID.

getGPL	logical, if TRUE, the corresponding GPL file will be downloaded. Default is TRUE.
update	logical, if TRUE, the previous stored Gene ExpressionSet RData will be updated. Default is FALSE

Value

Return an ExpressionSet class object.

Return an ExpressionSet class object.

Examples

```
## Not run:
# Download the GSE116028 which performed on GPL6480 platform
# from GEO and save it to the current directory
# Assign this ExpressionSet object to net_eset
net_eset <- load.exp.GEO(out.dir='./',
                         GSE='GSE116028',
                         GPL='GPL6480',
                         getGPL=TRUE,
                         update=FALSE)

## End(Not run)

## Not run:
# Download the GSE116028 which performed on GPL6480 platform
# from GEO and save it to the current directory
# Assign this ExpressionSet object to net_eset
net_eset <- load.exp.GEO(out.dir='./',
                         GSE='GSE116028',
                         GPL='GPL6480',
                         getGPL=TRUE,
                         update=FALSE)

## End(Not run)
```

---

load.exp.RNASeq.demo *Load Gene Expression Set from RNA-Seq Results (demo version)*

Description

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

load.exp.RNASeq.demo is a function to read in RNA-Seq results and convert it 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 = "tpm",
  merge_level = "gene"
)

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

Arguments

files	a vector of characters, the filenames for the transcript-level abundances. It will be passed to `tximport`. For details, please check `tximport`.
type	character, the type of software used to generate the abundances. It will be passed to `tximport`. For details, please check `tximport`.
tx2gene	data.frame or NULL, this parameter will be passed to `tximport`. For details, please check `tximport`.
use_phenotype_info	data.frame, the data frame contains phenotype information. It must have the columns `use_sample_col` and `use_design_col`.
use_sample_col	character, the column name, indicating which column in `use_phenotype_info` should be used as the sample name.
use_design_col	character, the column name, indicating which column in `use_phenotype_info` should be used as the design feature for samples.
return_type	character, the class of the return object. "txi" is the output of `tximport`. It is a list containing three matrices, abundance, counts and length. "counts" is the matrix of raw count. "tpm" is the raw tpm. "fpm", "cpm" is the fragments/counts per million mapped fragments. "raw-dds" is the DESeqDataSet class object, which is the original one without processing. "dds" is the DESeqDataSet class object, which is processed by `DESeq`. "eset" is the ExpressionSet class object, which is processed by `DESeq` and `vst`. Default is "tpm".
merge_level	character, users can choose between "gene" and "transcript". "gene", the original salmon results will be mapped to the transcriptome and the expression matrix will be merged to the gene level. This only works when using e.g. "gencode.vXX.transcripts.fa" from GENCODE as the reference.

Details

This function helps users to read in RNA-Seq results from various sources. Due to the complicated manipulations (e.g. reference sequence) in processing RNA-Seq, this demo function may not be suitable for all scenarios.

This function helps users to read in RNA-Seq results from various sources. Due to the complicated manipulations (e.g. reference sequence) in processing RNA-Seq, this demo function may not be suitable for all scenarios.

`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 it to eSet/DESeqDataSet class object.

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

Usage

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

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

Arguments

`salmon_dir`	character, the directory to save the results created by Salmon.
`tx2gene`	data.frame or NULL, this parameter will be passed to `tximport`. For details, please check `tximport`. If NULL, will read in one of the transcript names from Salmon's results. Note, it works when using e.g. "gencode.v29.transcripts.fa" from GENCODE as reference.
`use_phenotype_info`	data.frame, the data frame contains phenotype information. It must have the columns `use_sample_col` and `use_design_col`.
`use_sample_col`	character, the column name, indicating which column in `use_phenotype_info` should be used as the sample name.

use_design_col character, the column name, indicating which column in use_phenotype_info should be used as the design feature for samples.

return_type character, the class of the return object. "txi" is the output of tximport. It is a list containing three matrices, abundance, counts and length. "counts" is the matrix of raw count. "tpm" is the raw tpm. "fpm", "cpm" is the fragments/counts per million mapped fragments (fpm/cpm). "raw-dds" is the DESeqDataSet class object, which is the original one without processing. "dds" is the DESeqDataSet class object, which is processed by DESeq. "eset" is the ExpressionSet class object, which is processed by DESeq and vst. Default is "tpm".

merge_level character, users can choose between "gene" and "transcript". "gene", the original salmon results will be mapped to the transcriptome and the expression matrix will be merged to the gene level. This only works when using e.g. "gencode.vXX.transcripts.fa" from GENCODE as the reference.

Details

This function helps users to read in results created by Salmon. Due to the complicated manipulations (e.g. reference sequence) in processing Salmon, this demo function may not be suitable for all scenarios.

This function helps users to read in results created by Salmon. Due to the complicated manipulations (e.g. reference sequence) in processing Salmon, this demo function may not be suitable for all scenarios.

---

merge_eset *Merge Two ExpressionSet Class Objects into One*

Description

merge_eset merges two ExpressionSet class objects and returns one ExpresssionSet object. If genes in the two ExpressionSet objects are identical, the expression matrix will be combined directly. Otherwise, Z-transformation is strongly suggested to be performed before combination (set std=TRUE).

merge_eset merges two ExpressionSet class objects and returns one ExpresssionSet object. If genes in the two ExpressionSet objects are identical, the expression matrix will be combined directly. Otherwise, Z-transformation is strongly suggested to be performed before combination (set std=TRUE).

Usage

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

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

Arguments

eset1	ExpressionSet class, the first ExpressionSet.
eset2	ExpressionSet class, the second ExpressionSet.
group1	character, name of the first ExpressionSet.
group2	character, name of the second ExpressionSet.
group_col_name	character, name of the column which contains the names defined in group1 and group2. This column is designed to show which original ExpressionSet each sample comes from before combination. Default name of this column is "original_group".
use_col	a vector of characters, the column names in the phenotype information to be kept. If NULL, shared column names of eset1 and eset2 will be used. Default is NULL.
remove_batch	logical, if TRUE, remove the batch effects from these two expression datasets. Default is FALSE.
std	logical, whether to perform std to the original expression matrix. Default is FALSE.

Value

Return an ExressionSet class object.

Return an ExressionSet class 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)
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 Selected Major GeneSets from MsigDB*

Description

`merge_gs` combines selected major gene set collections (e.g. "H", "C1") together, and return a list object with sub-collections as elements. Each element contains a vector of genes belong to that sub-collection gene set.

`merge_gs` combines selected major gene set collections (e.g. "H", "C1") together, and return a list object with sub-collections as elements. Each element contains a vector of genes belong to that sub-collection gene set.

Usage

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

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

Arguments

`all_gs2gene`	list, the list returned by `gs.preload()`.
`use_gs`	a vector of characters, names of major gene set collections. Users can call `all_gs2gene_info` to see all the available collections. Default is c("H", "CP:BIOCARTA", "CP:REACTOME", "CP:KEGG").

Value

Return a list object with sub-collection gene sets as elements. Each element contains a vector of genes.

Return a list object with sub-collection gene sets as elements. Each element contains a vector of 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'))

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 Gene List for Two Drivers*

Description

merge_target_list merges target gene list for two drivers together. Shared target genes with high "MI (mutual information)" statistics will be kept in the final target list.

merge_target_list merges target gene list for two drivers together. Shared target genes with high "MI (mutual information)" statistics will be kept in the final target list.

Usage

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

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

Arguments

driver1	character, the name of the first driver.
driver2	character, the name of the second driver.
target_list	list, the driver-to-target list object. The names of the list elements are drivers (e.g. driver1 and driver2). Each element is a data frame, usually contains at least three columns. "target", target gene names; "MI", mutual information; "spearman", spearman correlation coefficient. Users can call get_net2target_list to create this list.

Value

Return a data.frame with rows of target genes, column of "target", "MI", "spearman".

Return a data.frame with rows of target genes, column of "target", "MI", "spearman".

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
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)
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
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 Values from TF (transcription factors) ExpressionSet Object and Sig (signaling factors) ExpressionSet Object*

Description

merge_TF_SIG.AC combines the activity value from TF (transcription factors) and Sig (signaling factors) ExpressionSet objects together, and adds "_TF" or "_SIG" suffix to drivers for easier distinction.

merge_TF_SIG.AC combines the activity value from TF (transcription factors) and Sig (signaling factors) ExpressionSet objects together, and adds "_TF" or "_SIG" suffix to drivers for easier distinction.

Usage

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

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

Arguments

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

Value

Return an ExpressionSet object.

Return an ExpressionSet object.

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,
                                            project_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)
## get eset (here for demo, use network.par$net.eset)
network.par <- list()
network.par$out.dir.DATA <- system.file('demo1','network/DATA/',package = "NetBID2")
NetBID.loadRDData(network.par=network.par,step='exp-QC')
analysis.par$cal.eset <- network.par$net.eset
ac_mat_TF <- cal.Activity(target_list=analysis.par$tf.network$target_list,
                      cal_mat=Biobase::exprs(analysis.par$cal.eset),
                      es.method='weightedmean')
ac_mat_SIG <- cal.Activity(target_list=analysis.par$tf.network$target_list,
                      cal_mat=Biobase::exprs(analysis.par$cal.eset),
```

```
                        es.method='weightedmean')
analysis.par$ac.tf.eset  <- generate.eset(exp_mat=ac_mat_TF,
                                  phenotype_info=Biobase::pData(analysis.par$cal.eset))
analysis.par$ac.sig.eset <- generate.eset(exp_mat=ac_mat_SIG,
                                  phenotype_info=Biobase::pData(analysis.par$cal.eset))
analysis.par$merge.ac.eset <- merge_TF_SIG.AC(TF_AC=analysis.par$ac.tf.eset,
                                      SIG_AC=analysis.par$ac.sig.eset)
if(exists('analysis.par')==TRUE) rm(analysis.par)
network.dir <- sprintf('%s/demo1/network/',system.file(package = "NetBID2")) # use demo
network.project.name <- 'project_2019-02-14' # demo project name
project_main_dir <- 'test/'
project_name <- 'test_driver'
analysis.par  <- NetBID.analysis.dir.create(project_main_dir=project_main_dir,
                                        project_name=project_name,
                                        network_dir=network.dir,
                                        network_project_name=network.project.name)
analysis.par$tf.network  <- get.SJAracne.network(network_file=analysis.par$tf.network.file)
analysis.par$sig.network <- get.SJAracne.network(network_file=analysis.par$sig.network.file)
## get eset (here for demo, use network.par$net.eset)
network.par <- list()
network.par$out.dir.DATA <- system.file('demo1','network/DATA/',package = "NetBID2")
NetBID.loadRData(network.par=network.par,step='exp-QC')
analysis.par$cal.eset <- network.par$net.eset
ac_mat_TF <- cal.Activity(target_list=analysis.par$tf.network$target_list,
                      cal_mat=Biobase::exprs(analysis.par$cal.eset),
                      es.method='weightedmean')
ac_mat_SIG <- cal.Activity(target_list=analysis.par$tf.network$target_list,
                      cal_mat=Biobase::exprs(analysis.par$cal.eset),
                      es.method='weightedmean')
analysis.par$ac.tf.eset  <- generate.eset(exp_mat=ac_mat_TF,
                                  phenotype_info=Biobase::pData(analysis.par$cal.eset))
analysis.par$ac.sig.eset <- generate.eset(exp_mat=ac_mat_SIG,
                                  phenotype_info=Biobase::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 TF (transcription factor) Network and Sig (signaling factor) Network*

Description

`merge_TF_SIG.network` takes TF network and Sig network and combine them together. The merged list object contains three elements, a data.frame contains all the combined network information `network_dat`, a driver-to-target list object `target_list`, and an igraph object of the network `igraph_obj`.

`merge_TF_SIG.network` takes TF network and Sig network and combine them together. The merged list object contains three elements, a data.frame contains all the combined network information `network_dat`, a driver-to-target list object `target_list`, and an igraph object of the network `igraph_obj`.

Usage

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

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

Arguments

TF_network	list, the TF network created by get.SJAracne.network function.
SIG_network	list, the SIG network created by get.SJAracne.network function.

Value

Return the a list containing three elements, network_dat, target_list and igraph_obj.

Return the a list containing three elements, 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,
                                            project_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$sig.network)
if(exists('analysis.par')==TRUE) rm(analysis.par)
network.dir <- sprintf('%s/demo1/network/',system.file(package = "NetBID2")) # use demo
network.project.name <- 'project_2019-02-14' # demo project name
project_main_dir <- 'test/'
project_name <- 'test_driver'
analysis.par  <- NetBID.analysis.dir.create(project_main_dir=project_main_dir,
                                            project_name=project_name,
                                            network_dir=network.dir,
                                            network_project_name=network.project.name)
analysis.par$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

Manipulation of Working Directories for NetBID2 Driver Estimation Step

---

Description

NetBID.analysis.dir.create is used to help users create an organized working directory for the driver estimation step in NetBID2 analysis. However, it is not essential for the analysis. It creates a hierarchcial working directory and returns a list contains this directory information.

NetBID.analysis.dir.create is used to help users create an organized working directory for the driver estimation step in NetBID2 analysis. However, it is not essential for the analysis. It creates a hierarchcial working directory and returns a list contains this directory information.

Usage

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

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

Arguments

project_main_dir	character, name or absolute path of the main working directory for driver analysis.
project_name	character, name of the project folder.
network_dir	character, name or absolute path of the main working directory for network construction.
network_project_name	character, the project name of network construction. Or use the project name of SJARACNe. This parameter is optional. If one didn't run NetBID2 network construction part in the pipeline, he could set it to NULL. If one like to follow the NetBID2 pipeline, he should set it to the path of the TF network file and the SIG network file.
tf.network.file	character, the path of the TF network file (e.g. "XXX/consensus_network_ncol_.txt"). Default is the path of network_project_name.
sig.network.file	character, the path of the SIG network file (e.g. "XXX/consensus_network_ncol_.txt"). Default is the path of network_project_name.

Details

This function requires user to define the main working directory and the project's name. It creates a main working directory with a subdirectory of the project. It also automatically creates three subfolders (QC, DATA and PLOT) within the project folder. QC/, storing Quality Control related plots; DATA/, saving data in RData format; PLOT/, storing output plots. This function also returns a list object (e.g. `analysis.par` in the demo) with directory information wrapped inside. This list is an essential for driver construction step, all the important intermediate data generated later will be wrapped inside.

This function requires user to define the main working directory and the project's name. It creates a main working directory with a subdirectory of the project. It also automatically creates three subfolders (QC, DATA and PLOT) within the project folder. QC/, storing Quality Control related plots; DATA/, saving data in RData format; PLOT/, storing output plots. This function also returns a list object (e.g. `analysis.par` in the demo) with directory information wrapped inside. This list is an essential for driver construction step, all the important intermediate data generated later will be wrapped inside.

Value

Returns a list object, containing main.dir (path of the main working directory), project.name (project name), out.dir (path of the project folder, which contains three subfolders), out.dir.QC, out.dir.DATA and out.dir.PLOT.

Returns a list object, containing main.dir (path of the main working directory), project.name (project name), out.dir (path of the project folder, which contains three subfolders), out.dir.QC, out.dir.DATA and 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,
                                            project_name=project_name,
                                            network_dir=network.dir,
                                            network_project_name=network.project.name)

## End(Not run)

## 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,
                                            project_name=project_name,
                                            network_dir=network.dir,
                                            network_project_name=network.project.name)

## End(Not run)
```

NetBID.lazyMode.DriverEstimation
Lazy mode for NetBID2 driver estimation

Description

NetBID.lazyMode.DriverEstimation is an integrated function for NetBID2 driver estimation.

NetBID.lazyMode.DriverEstimation is an integrated function for NetBID2 driver estimation.

Usage

```
NetBID.lazyMode.DriverEstimation(
  project_main_dir = NULL,
  project_name = NULL,
  tf.network.file = NULL,
  sig.network.file = NULL,
  cal.eset = NULL,
  main_id_type = NULL,
  cal.eset_main_id_type = NULL,
  use_level = "gene",
  transfer_tab = NULL,
  intgroup = NULL,
  G1_name = NULL,
  G0_name = NULL,
  comp_name = NULL,
  do.QC = TRUE,
  DE_strategy = "bid",
  return_analysis.par = TRUE
)

NetBID.lazyMode.DriverEstimation(
  project_main_dir = NULL,
  project_name = NULL,
  tf.network.file = NULL,
  sig.network.file = NULL,
  cal.eset = NULL,
  main_id_type = NULL,
  cal.eset_main_id_type = NULL,
  use_level = "gene",
  transfer_tab = NULL,
  intgroup = NULL,
  G1_name = NULL,
  G0_name = NULL,
  comp_name = NULL,
  do.QC = TRUE,
  DE_strategy = "bid",
  return_analysis.par = TRUE
)
```

Arguments

project_main_dir	character, name or absolute path of the main working directory for driver analysis.
project_name	character, name of the project folder.
tf.network.file	character, the path of the TF network file (e.g. "XXX/consensus_network_ncol_.txt").
sig.network.file	character, the path of the SIG network file (e.g. "XXX/consensus_network_ncol_.txt").
cal.eset	ExpressionSet class, the ExpressionSet for analysis.
main_id_type	character, the type of driver's ID. It comes from the attribute name in biomaRt package. Such as "ensembl_gene_id", "ensembl_gene_id_version", "ensembl_transcript_id", "ensembl_transcript_id_version" or "refseq_mrna". For details, user can call `biomaRt::listAttributes()` to display all available attributes in the selected dataset.
cal.eset_main_id_type	character, the type of cal.eset's ID. It comes from the attribute name in biomaRt package. Such as "ensembl_gene_id", "ensembl_gene_id_version", "ensembl_transcript_id", "ensembl_transcript_id_version" or "refseq_mrna". For details, user can call `biomaRt::listAttributes()` to display all available attributes in the selected dataset.
use_level	character, users can choose "transcript" or "gene". Default is "gene".
transfer_tab	data.frame, the ID conversion table. Users can call `get_IDtransfer` to get this table. Only useful when "cal.eset_main_id_type" does not equal to "main_id_type". This is mainly for converting ID for cal.eset, must include "cal.eset_main_id_type" and "main_id_type". If NULL, will automatically generate it.
intgroup	character, one interested phenotype group from the `cal.eset`.
G1_name	character, the name of experimental group (e.g. "Male"), must be the character in `intgroup`.
G0_name	character, the name of control group (e.g. "Female"), must be the character in `intgroup`.
comp_name	character, the name of the comparison of interest.
do.QC	logical, if TRUE, will perform network QC and activity eSet QC plots. Default is TRUE.
DE_strategy	character, use limma or bid to calculate differentiated expression/activity. Default is 'bid'.
return_analysis.par	logical, if TRUE, will return the complicated list object analysis.par.

Details

The function will return the complicated list object analysis.par if set return_analysis.par=TRUE. Meanwhile, the master table and the RData containing analysis.par will be automatically saved.

The function will return the complicated list object analysis.par if set return_analysis.par=TRUE. Meanwhile, the master table and the RData containing analysis.par will be automatically saved.

Examples

```
## Not run:
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'
tf.network.file <- sprintf('%s/SJAR/%s/output_tf_sjaracne_%s_out_.final/%s',
                network.dir,network.project.name,network.project.name,
                'consensus_network_ncol_.txt')
sig.network.file <- sprintf('%s/SJAR/%s/output_sig_sjaracne_%s_out_.final/%s',
                network.dir,network.project.name,network.project.name,
                'consensus_network_ncol_.txt')
load(sprintf('%s/DATA/network.par.Step.exp-QC.RData',network.dir))
cal.eset <- network.par$net.eset
db.preload(use_level='gene')
analysis.par <- NetBID.lazyMode.DriverEstimation(project_main_dir=project_main_dir,
                project_name=project_name,
                tf.network.file=tf.network.file,
                sig.network.file=sig.network.file,
                cal.eset=cal.eset,
                main_id_type='external_gene_name',
                cal.eset_main_id_type='external_gene_name',
                intgroup='subgroup',
                G1_name='G4',G0_name='SHH',
                comp_name='G4.Vs.SHH',
                do.QC=FALSE,return_analysis.par=TRUE)

## End(Not run)
## Not run:
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'
tf.network.file <- sprintf('%s/SJAR/%s/output_tf_sjaracne_%s_out_.final/%s',
                network.dir,network.project.name,network.project.name,
                'consensus_network_ncol_.txt')
sig.network.file <- sprintf('%s/SJAR/%s/output_sig_sjaracne_%s_out_.final/%s',
                network.dir,network.project.name,network.project.name,
                'consensus_network_ncol_.txt')
load(sprintf('%s/DATA/network.par.Step.exp-QC.RData',network.dir))
cal.eset <- network.par$net.eset
db.preload(use_level='gene')
analysis.par <- NetBID.lazyMode.DriverEstimation(project_main_dir=project_main_dir,
                project_name=project_name,
                tf.network.file=tf.network.file,
                sig.network.file=sig.network.file,
                cal.eset=cal.eset,
                main_id_type='external_gene_name',
                cal.eset_main_id_type='external_gene_name',
                intgroup='subgroup',
                G1_name='G4',G0_name='SHH',
                comp_name='G4.Vs.SHH',
                do.QC=FALSE,return_analysis.par=TRUE)

## End(Not run)
```

NetBID.lazyMode.DriverVisualization
Lazy mode for NetBID2 result visualization

Description

NetBID.lazyMode.DriverVisualization is an integrated function to draw visualization plots for top drivers.

NetBID.lazyMode.DriverVisualization is an integrated function to draw visualization plots for top drivers.

Usage

```
NetBID.lazyMode.DriverVisualization(
  analysis.par = NULL,
  intgroup = NULL,
  use_comp = NULL,
  main_id_type = "external_gene_name",
  transfer_tab = NULL,
  use_gs = c("H", "CP:BIOCARTA", "CP:REACTOME", "CP:KEGG"),
  min_Size = 30,
  max_Size = 1000,
  top_number = 30,
  top_strategy = "Both",
  logFC_thre = 0.05,
  Pv_thre = 0.05
)

NetBID.lazyMode.DriverVisualization(
  analysis.par = NULL,
  intgroup = NULL,
  use_comp = NULL,
  main_id_type = "external_gene_name",
  transfer_tab = NULL,
  use_gs = c("H", "CP:BIOCARTA", "CP:REACTOME", "CP:KEGG"),
  min_Size = 30,
  max_Size = 1000,
  top_number = 30,
  top_strategy = "Both",
  logFC_thre = 0.05,
  Pv_thre = 0.05
)
```

Arguments

analysis.par	list, stores all related datasets from driver analysis step.
intgroup	character, one interested phenotype group from the analysis.par$cal.eset.
use_comp	character, the name of the comparison of interest, should be included in the colnames of analysis.par$DA and analysis.par$DE.

main_id_type	character, the type of driver's ID. It comes from the attribute name in biomaRt package. Such as "ensembl_gene_id", "ensembl_gene_id_version", "ensembl_transcript_id", "ensembl_transcript_id_version" or "refseq_mrna". For details, user can call `biomaRt::listAttributes()` to display all available attributes in the selected dataset.
transfer_tab	data.frame, the ID conversion table. Users can call `get_IDtransfer` to get this table.
use_gs	a vector of characters, names of major gene set collections. Users can call `all_gs2gene_info` to see all the available collections. Default is c("H", "CP:BIOCARTA", "CP:REACTOME", "CP:KEGG").
min_Size	numeric, minimum size for the target genes. Default is 30.
max_Size	numeric, maximum size for the target genes. Default is 1000.
top_number	number for the top significant genes/drivers in the combine results to be displayed on the plot. Default is 30.
top_strategy	character, choose from "Both", "Up", "Down". If set to "Both", top drivers with highest absolute Z statistics will be displayed. If set to "Up", only top up-regulated drivers will be displayed. If set to "Down", only top down-regulated drivers will be displayed. Default is "Both".
logFC_thre	numeric, the threshold of logFC. Genes or drivers with absolute logFC value higher than the threshold will be kept. Default is 0.05.
Pv_thre	numeric, the threshold of P-values. Genes or drivers with P-values lower than the threshold will be kept. Default is 0.05.

Details

User need to strictly follow the NetBID2 pipeline to get the complicated list object analysis.par. "intgroup", "use_comp" should be specified; "transfer_tab" could be set to NULL with "main_id_type" specified, but it is suggested to input by hand if available.

User need to strictly follow the NetBID2 pipeline to get the complicated list object analysis.par. "intgroup", "use_comp" should be specified; "transfer_tab" could be set to NULL with "main_id_type" specified, but it is suggested to input by hand if available.

Examples

```
## Not run:
analysis.par <- list()
analysis.par$out.dir.DATA <- system.file('demo1','driver/DATA/',package = "NetBID2")
NetBID.loadRData(analysis.par=analysis.par,step='ms-tab')
analysis.par$out.dir.PLOT <- 'test/'
NetBID.lazyMode.DriverVisualization(analysis.par=analysis.par,
                                    intgroup='subgroup',use_comp='G4.Vs.others',
                                    transfer_tab=analysis.par$transfer_tab,
                                    logFC_thre=0.2,Pv_thre=1e-4)

## End(Not run)
## Not run:
analysis.par <- list()
analysis.par$out.dir.DATA <- system.file('demo1','driver/DATA/',package = "NetBID2")
NetBID.loadRData(analysis.par=analysis.par,step='ms-tab')
analysis.par$out.dir.PLOT <- 'test/'
NetBID.lazyMode.DriverVisualization(analysis.par=analysis.par,
                                    intgroup='subgroup',use_comp='G4.Vs.others',
```

```
                                transfer_tab=analysis.par$transfer_tab,
                                logFC_thre=0.2,Pv_thre=1e-4)

## End(Not run)
```

NetBID.loadRData *Reload Saved RData Created by* NetBID.saveRData.

Description

NetBID.loadRData is a function loads RData saved by NetBID.saveRData function. It prevents user from repeating former pipeline steps.

NetBID.loadRData is a function loads RData saved by NetBID.saveRData function. It prevents user from repeating former pipeline steps.

Usage

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

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

Arguments

network.par	list, stores all related datasets from network construction step.
analysis.par	list, stores all related datasets from driver analysis step.
step	character, name of the pipeline step. It should be previously assigned by user when calling NetBID.saveRData function.

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 <- 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

Manipulation of Working Directories for NetBID2 Network Construction Step

---

Description

NetBID.network.dir.create is used to help users create an organized working directory for the network construction step in NetBID2 analysis. However, it is not essential for the analysis. It creates a hierarchcial working directory and returns a list contains this directory information.

NetBID.network.dir.create is used to help users create an organized working directory for the network construction step in NetBID2 analysis. However, it is not essential for the analysis. It creates a hierarchcial working directory and returns a list contains this directory information.

Usage

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

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

Arguments

project_main_dir	character, name or absolute path of the main working directory.
project_name	character, name of the project folder.

Details

This function needs users to define the main working directory and the project's name. It creates a main working directory with a subdirectory of the project. It also automatically creates three subfolders (QC, DATA and SJAR) within the project folder. QC/, storing Quality Control related plots; DATA/, saving data in RData format; SJAR/, storing files needed for running SJAracne command. This function also returns a list object (example, `network.par` in the demo) with directory information wrapped inside. This list is an essential for network construction step, all the important intermediate data generated later will be wrapped inside.

This function needs users to define the main working directory and the project's name. It creates a main working directory with a subdirectory of the project. It also automatically creates three subfolders (QC, DATA and SJAR) within the project folder. QC/, storing Quality Control related plots; DATA/, saving data in RData format; SJAR/, storing files needed for running SJAracne command. This function also returns a list object (example, `network.par` in the demo) with directory information wrapped inside. This list is an essential for network construction step, all the important intermediate data generated later will be wrapped inside.

Value

`NetBID.network.dir.create` returns a list object, containing main.dir (path of the main working directory), project.name (project name), out.dir (path of the project folder, which contains three subfolders), out.dir.QC, out.dir.DATA and out.dir.SJAR.

`NetBID.network.dir.create` returns a list object, containing main.dir (path of the main working directory), project.name (project name), out.dir (path of the project folder, which contains three subfolders), out.dir.QC, out.dir.DATA and out.dir.SJAR.

Examples

```
## Not run:
# Creating a main working directory under the current working directory by folder name
network.par <- NetBID.network.dir.create("MyMainDir","MyProject")
# Or creating a main working directory under the current working directory by relative path
network.par <- NetBID.network.dir.create("./MyMainDir","MyProject")
# Or creating a main working directory to a specific path by absolute path
network.par <- NetBID.network.dir.create("~/Desktop/MyMainDir","MyProject")

## End(Not run)

## Not run:
# Creating a main working directory under the current working directory by folder name
network.par <- NetBID.network.dir.create("MyMainDir","MyProject")
# Or creating a main working directory under the current working directory by relative path
```

```
network.par <- NetBID.network.dir.create("./MyMainDir","MyProject")
# Or creating a main working directory to a specific path by absolute path
network.par <- NetBID.network.dir.create("~/Desktop/MyMainDir","MyProject")

## End(Not run)
```

`NetBID.saveRData` *Save Data Produced by Corresponding NetBID2 Pipeline Step.*

Description

`NetBID.saveRData` is a function to save complicated list object generated by certain steps of NetBID2's pipeline (e.g. load gene expression file from GEO, 'exp-load'). This function is not essential, but it is highly suggested for easier pipeline step checkout and reference.

`NetBID.saveRData` is a function to save complicated list object generated by certain steps of NetBID2's pipeline (e.g. load gene expression file from GEO, 'exp-load'). This function is not essential, but it is highly suggested for easier pipeline step checkout and reference.

Usage

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

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

Arguments

`network.par`	list, stores all related datasets from network construction pipeline step.
`analysis.par`	list, stores all related datasets from driver analysis pipeline step.
`step`	character, name of the pipeline step decided by user for easier reference.

Details

There are two important steps in the NetBID2 pipeline, network construction and driver analysis. User can save these two complicated list objects, network.par and analysis.par. Assigning the `step` name to save the RData for easier reference. Calling `NetBID.loadRData` to load the corresponding step RData, users can avoid repeating the former steps.

There are two important steps in the NetBID2 pipeline, network construction and driver analysis. User can save these two complicated list objects, network.par and analysis.par. Assigning the `step` name to save the RData for easier reference. Calling `NetBID.loadRData` to load the corresponding step RData, users can avoid 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)
## 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` *Save the Master Table into Excel File*

Description

`out2excel` is a function can save data frame as Excel File. This is mainly for the output of master table generated by `generate.masterTable`.

`out2excel` is a function can save data frame as Excel File. This is mainly for the output of 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
)

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 data.frame, if data.frame, it is generated by `generate.masterTable`. If list, each list element is data.frame/master table. The name of the list element will be the sheet name in the excel file.
`out.xlsx`	character, path and file name of the output Excel file.
`mark_gene`	list, list of marker genes. The name of the list element is the marked group name. Each element is a vector of marker genes. This is optional, just to add additional information to the file.

mark_col	character, the color to mark the marker genes. If NULL, will use get.class.color to get the colors.
mark_strategy	character, users can choose between "color" and "add_column". "Color" means the mark_gene will be marked by filling its background color; "add_column" means the mark_gene will be displayed in a separate column with TRUE/FALSE, indicating whether the gene belongs to a mark group or not.
workbook_name	character, name of the workbook for the output Excel. Default is "ms_tab".
only_z_sheet	logical, if TRUE, will create a separate sheet only contains Z-statistics related columns from DA/DE analysis. Default is FALSE.
z_column	character, name of the columns contain Z-statistics. If NULL, find column names start with "Z.". Default is NULL.
sig_thre	numeric, threshold for the Z-statistics. Z values passed the threshold will be colored. The color scale is defined by z2col. Default is 1.64.

Value

Return a logical value. If TRUE, the Excel file has been generated successfully.

Return a logical value. If TRUE, the Excel file has been generated successfully.

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),
            pre_define=c('WNT'='blue','SHH'='red',
            'Group3'='yellow','Group4'='green'))
outfile <- 'test_out.xlsx'
out2excel(ms_tab,out.xlsx = outfile,mark_gene,mark_col)

## End(Not run)
## 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),
            pre_define=c('WNT'='blue','SHH'='red',
            'Group3'='yellow','Group4'='green'))
outfile <- 'test_out.xlsx'
out2excel(ms_tab,out.xlsx = outfile,mark_gene,mark_col)

## End(Not run)
```

`processDriverProfile` *Clean Activity-based profile*

Description

`processDriverProfile` is a helper function to pre-process Activity-based profile.

`processDriverProfile` is a helper function to pre-process Activity-based profile.

Usage

```
processDriverProfile(
  Driver_profile,
  Driver_name,
  choose_strategy = "min",
  return_type = "driver_name"
)

processDriverProfile(
  Driver_profile,
  Driver_name,
  choose_strategy = "min",
  return_type = "driver_name"
)
```

Arguments

`Driver_profile`	a numeric vector, contain statistics for drivers (e.g driver's target size, driver's Z-statistics)
`Driver_name`	a character vector, contain name for drivers. The length of 'Driver_profile' and 'Driver_name' must be equal and the order of item must match.
`choose_strategy`	character, strategy of selection if duplicate driver name (e.g TP53_TF, TP53_SIG). Choose from "min","max","absmin","absmax". Default is "min".
`return_type`	character, strategy of return type. Choose from "driver_name","gene_name", "driver_statistics", "gene_statistics". If choose "*_name", only the name vector is returned. If choose "*_statistics", the statistics vector is returned with character name. Default is "driver_name".

Examples

```
analysis.par <- list()
analysis.par$out.dir.DATA <- system.file('demo1','driver/DATA/',package = "NetBID2")
NetBID.loadRData(analysis.par=analysis.par,step='ms-tab')
ms_tab <- analysis.par$final_ms_tab
Driver_profile <- ms_tab$P.Value.G4.Vs.WNT_DA
Driver_name <- ms_tab$gene_label
res1 <- processDriverProfile(Driver_profile=Driver_profile,
                             Driver_name=Driver_name,
                             choose_strategy='min')
res2 <- processDriverProfile(Driver_profile=Driver_profile,
                             Driver_name=Driver_name,
```

```
                              return_type = 'gene_name',
                              choose_strategy='min')
Driver_profile <- ms_tab$Z.G4.Vs.WNT_DA
res3 <- processDriverProfile(Driver_profile=Driver_profile,
                              Driver_name=Driver_name,
                              choose_strategy='absmax',
                              return_type = 'driver_statistics')
res4 <- processDriverProfile(Driver_profile=Driver_profile,
                              Driver_name=Driver_name,
                              choose_strategy='absmax',
                              return_type = 'gene_statistics')
driver_size <- ms_tab$Size
res5 <- processDriverProfile(Driver_profile=driver_size,
                              Driver_name=Driver_name,
                              choose_strategy='max')
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
Driver_profile <- ms_tab$P.Value.G4.Vs.WNT_DA
Driver_name <- ms_tab$gene_label
res1 <- processDriverProfile(Driver_profile=Driver_profile,
                              Driver_name=Driver_name,
                              choose_strategy='min')
res2 <- processDriverProfile(Driver_profile=Driver_profile,
                              Driver_name=Driver_name,
                              return_type = 'gene_name',
                              choose_strategy='min')
Driver_profile <- ms_tab$Z.G4.Vs.WNT_DA
res3 <- processDriverProfile(Driver_profile=Driver_profile,
                              Driver_name=Driver_name,
                              choose_strategy='absmax',
                              return_type = 'driver_statistics')
res4 <- processDriverProfile(Driver_profile=Driver_profile,
                              Driver_name=Driver_name,
                              choose_strategy='absmax',
                              return_type = 'gene_statistics')
driver_size <- ms_tab$Size
res5 <- processDriverProfile(Driver_profile=driver_size,
                              Driver_name=Driver_name,
                              choose_strategy='max')
```

`RNASeqCount.normalize.scale`

Normalization of RNA-Seq Reads Count

Description

`RNASeqCount.normalize.scale` is a simple version to normalize the RNASeq reads count data.

`RNASeqCount.normalize.scale` is a simple version to normalize the RNASeq reads count data.

Usage

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

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

Arguments

mat	matrix, matrix of RNA-Seq reads data. Each row is a gene/transcript, each column is a sample.
total	integer, total RNA-Seq reads count. If NULL, will use the mean of each column's summation. Default is NULL.
pseudoCount	integer, the integer added to avoid "-Inf" showing up during log transformation. Default is 1.

Details

Users can also load `load.exp.RNASeq.demo`, and follow the `DESeq2` pipeline for RNASeq data processing. Warning, `load.exp.RNASeq.demo` and `load.exp.RNASeq.demoSalmon` in NetBID2 may not cover all the possible scenarios.

Users can also load `load.exp.RNASeq.demo`, and follow the `DESeq2` pipeline for RNASeq data processing. Warning, `load.exp.RNASeq.demo` and `load.exp.RNASeq.demoSalmon` in NetBID2 may not cover all the possible scenarios.

Value

Return a numeric matrix, containing the normalized RNA-Seq reads count.

Return a numeric matrix, containing the normalized RNA-Seq reads 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)
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 Data Files for Running SJARACNe*

Description

`SJAracne.prepare` prepares data files for running SJAracne. SJARACNe is a scalable software tool for gene network reverse engineering from big data. Detailed description and how to run SJARACNe can be found in its GitHub repository. The usage of SJARACNe may be updated and the bash file generated by this function may not fit the version in use (not suit for SJAracne 0.2.0). Please check https://github.com/jyyulab/SJARACNe/ for details.

`SJAracne.prepare` prepares data files for running SJAracne. SJARACNe is a scalable software tool for gene network reverse engineering from big data. Detailed description and how to run SJARACNe can be found in its GitHub repository. The usage of SJARACNe may be updated and the bash file generated by this function may not fit the version in use (not suit for SJAracne 0.2.0). Please check https://github.com/jyyulab/SJARACNe/ for details.

Usage

```
SJAracne.prepare(
  eset,
  use.samples = rownames(Biobase::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 = "",
  geneSymbol_column = NULL
)

SJAracne.prepare(
  eset,
  use.samples = rownames(Biobase::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 = "",
  geneSymbol_column = NULL
)
```

Arguments

eset	an ExpressionSet class object, which contains the expression matrix.
use.samples	a vector of characters, the list of sample used to run SJARACNe.
TF_list	a vector of characters, the TF list.
SIG_list	a vector of characters, the SIG list.
SJAR.main_dir	character, the path to save the results generated by SJARACNe.
SJAR.project_name	character, the project name used to label the output directory.
IQR.thre	numeric, the IQR filter threshold to filter all non-driver genes.
IQR.loose_thre	numeric, the IQR filter threshold to filter for all driver(TF/SIG) genes.
add_options	additional option for running SJARACNe.
geneSymbol_column	character, the column name in fdata(eset) which contains gene symbol. If NULL, will use the main ID in exprs(eset) to fulfill the "geneSymbol" column. Default is NULL.

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(Biobase::fData(network.par$net.eset))
use_list  <- get.TF_SIG.list(use_genes,use_gene_type=use_gene_type)
#select sample for analysis
phe <- Biobase::pData(network.par$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)
## Not run:
network.par <- list()
network.par$out.dir.DATA <- system.file('demo1','network/DATA/',package = "NetBID2")
NetBID.loadRData(network.par=network.par,step='exp-QC')
db.preload(use_level='gene',use_spe='human',update=FALSE)
use_gene_type <- 'external_gene_name' ## this should user-defined !!!
use_genes <- rownames(Biobase::fData(network.par$net.eset))
use_list  <- get.TF_SIG.list(use_genes,use_gene_type=use_gene_type)
#select sample for analysis
phe <- Biobase::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 Intersection of Target Genes between Two Drivers*

Description

`test.targetNet.overlap` performs Fisher's exact test to see whether the target genes from two drivers are significantly intersected.

`test.targetNet.overlap` performs Fisher's exact test to see whether the target genes from two drivers are significantly intersected.

Usage

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

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

Arguments

source1_label	character, the label of the first selected driver.
source2_label	character, the label of the second selected driver.
target1	a vector of characters, the list of target genes from the first driver.
target2	a vector of characters, the list of target genes from the second driver.
total_possible_target	numeric or a vector of characters. If input is numeric, it is the total number of possible target genes. If input is a vector of characters, it is the background list of all possible target genes.

Value

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

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

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:

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` *Reassign featureData slot of ExpressionSet and Update feature information*

Description

`update_eset.feature` reassigns the featureData slot of ExpressionSet object based on user's demand. It is mainly used for gene ID conversion.

`update_eset.feature` reassigns the featureData slot of ExpressionSet object based on user's demand. It is mainly used for gene ID conversion.

Usage

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

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.
`use_feature_info`	data.frame, a data frame contains feature information, it can be obtained by calling `fData` function.
`from_feature`	character, orginal ID. Must be one of the column names in `use_feature_info` and correctly characterize the `use_eset`'s row names.
`to_feature`	character, target ID. Must be one of the column names in `use_feature_info`.
`merge_method`	character, the agglomeration method to be used for merging gene expression value. This should be one of, "median", "mean", "max" or "min". Default is "median".
`distribute_method`	character, the agglomeration method to be used for distributing the gene expression value. This should be one of, "mean" or "equal". Default is "equal".

Details

User can pass a conversion table to use_eset for the ID conversion. A conversion table can be obtained from the original featureDta slot (if one called the load.exp.GEO function and set getGPL==TRUE) or by running the get_IDtransfer function. The mapping between original ID and target ID can be summerised into 4 categories. 1) One-to-one, simply replaces the original ID with target ID; 2) Many-to-one, the expression value for the target ID will be merged from its original ID; 3) One-to-many, the expression value for the original ID will be distributed to the matched target IDs; 4) Many-to-many, apply part 3) first, then part 2).

User can pass a conversion table to use_eset for the ID conversion. A conversion table can be obtained from the original featureDta slot (if one called the load.exp.GEO function and set getGPL==TRUE) or by running the get_IDtransfer function. The mapping between original ID and target ID can be summerised into 4 categories. 1) One-to-one, simply replaces the original ID with target ID; 2) Many-to-one, the expression value for the target ID will be merged from its original ID; 3) One-to-many, the expression value for the original ID will be distributed to the matched target IDs; 4) Many-to-many, apply part 3) first, then part 2).

Value

Return an ExressionSet object with updated feature information.

Return an ExressionSet object with updated feature information.

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(Biobase::exprs(eset)[test_transfer_table$Gene,])
print(Biobase::exprs(new_eset))

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(Biobase::exprs(eset)[test_transfer_table$Gene,])
print(Biobase::exprs(new_eset))
```

`update_eset.phenotype` *Reassign the phenoData slot of ExpressionSet and Update phenotype information*

Description

`update_eset.phenotype` reassigns the phenoData slot of ExpressionSet based on user's demand. It is mainly used to modify sample names and extract interested phenotype information for further sample clustering.

`update_eset.phenotype` reassigns the phenoData slot of ExpressionSet based on user's demand. It is mainly used to modify sample names and extract interested phenotype information for further sample clustering.

Usage

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

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

Arguments

`use_eset`	ExpressionSet class object.
`use_phenotype_info`	data.frame, a dataframe contains phenotype information, can be obtained by calling `pData` function.
`use_sample_col`	character, must be one of the column names in `use_phenotype_info`.
`use_col`	character, the columns will be kept from `use_phenotype_info`. 'auto', only extracting 'cluster-meaningful' sample features (e.g. it is meaningless to use 'gender' as clustering feature, if all samples are female). 'GEO-auto' means it will extract the following selected columns, "geo_accession", "title", "source_name_ch1", and columns ended with ":ch1". Default is "auto".

Value

Return an ExressionSet object with updated phenotype information.

Return an ExressionSet object with updated phenotype information.

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=Biobase::pData(net_eset),
                                  use_sample_col='geo_accession',
                                  use_col='GEO-auto')

## End(Not run)
## 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=Biobase::pData(net_eset),
                                  use_sample_col='geo_accession',
                                  use_col='GEO-auto')

## End(Not run)
```

`update_SJAracne.network`

Update Network List Object Using Constraints

Description

`update_SJAracne.network` updates the network object created by `get.SJAracne.network`, using constraints like statistical thresholds and interested gene list.

`update_SJAracne.network` updates the network object created by `get.SJAracne.network`, using constraints like statistical thresholds and interested gene list.

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
)

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 object created by get.SJAracne.network. The list contains three elements, network_dat, target_list and igraph_obj. For details, please check get.SJAracne.network.
all_possible_drivers	a vector of characters, all possible drivers used to filter the network. If NULL, will use drivers from network_list. Default is NULL.
all_possible_targets	a vector of characters, all possible target genes used to filter the network. If NULL, will use targets from network_list. Default is NULL.
force_all_drivers	logical, if TRUE, will include all drivers from all_possible_drivers into the final network. For network_dat and target_list in the network list object, all genes in all_possible_drivers will not be filtered using the following statistical thresholds. For igraph_obj in the network list object, all genes in all_possible_drivers that don't exist in the original network, will be kept as vertices. Default is TRUE.
force_all_targets	logical, if TRUE, will include all genes from all_possible_targets into the final network. For network_dat and target_list in the network list object, all genes in all_possible_drivers will not be filtered using the following statistical thresholds. For igraph_obj in the network list object, all genes in all_possible_drivers that don't exist in the original network, will be kept as vertices. Default is TRUE.
min_MI	numeric, minimum threshold for "MI (mutual information)". 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 numerics, users can choose from 1, -1 and c(1, -1). 1 means only positve spearman values will be used. -1 means only negative spearman values will be used. Default is c(1,-1).

pearson_sign_use
: a vector of numerics, users can choose from 1, -1 and c(1, -1). 1 means only positve pearson values will be used. -1 means only negative pearson values will be used. Default is c(1,-1).

directed
: logical, if TRUE, the network is a directed graph. Default is TRUE.

weighted
: logical, if TRUE, the network is weighted. Default is TRUE.

Value

Return a list containing three elements, network_dat, target_list and igraph_obj.

Return a list containing three elements, 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,
                                            project_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$tf.network,
                                             all_possible_drivers=all_possible_drivers,
                                             force_all_drivers=TRUE,
                                             force_all_targets=FALSE,
                                             pearson_sign_use=1)
print(base::intersect(c('addition_driver_1','addition_driver_2'),
      names(V(tf.network.update$igraph_obj)))) ## check
## 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,
                                            project_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(base::intersect(c('addition_driver_1','addition_driver_2'),
      names(V(tf.network.update$igraph_obj)))) ## check
## Not run:
```

---

z2col *Set Color Scale for Z Statistics Value*

Description

z2col is a helper function in out2excel. It defines the color scale of the Z statistics value.

z2col is a helper function in out2excel. It defines the color scale of the Z statistics value.

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]
)

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 numerics, a vector of Z statistics.
n_len	integer, number of unique colors. Default is 60.
sig_thre	numeric, the threshold for significance (absolute value of Z statistics). Z values failed to pass the threshold will be colored "white".
col_min_thre	numeric, the lower threshold for the color bar value. Default is 0.01.
col_max_thre	numeric, the upper threshold for the color bar value. Default is 3.
blue_col	a vector of characters, the blue colors used to show the negative Z values. Default is brewer.pal(9,'Set1')[2].
red_col	a vector of characters, the red colors used to show positive Z values. Default is brewer.pal(9,'Set1')[1].

Value

Return a vector of color codes.

Return a vector of color codes.

Examples

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

Index