Package ‘EnrichmentBrowser’

April 1, 2024

Version 2.32.0
Date 2023-07-29
Title Seamless navigation through combined results of set-based and network-based enrichment analysis
Depends SummarizedExperiment, graph
Imports AnnotationDbi, BiocFileCache, BiocManager, GSEABase, GO.db, KEGGREST, KEGGgraph, Rgraphviz, S4Vectors, SPIA, edgeR, graphite, hwriter, limma, methods, pathview, safe
Suggests ALL, BiocStyle, ComplexHeatmap, DESeq2, ReportingTools, airway, biocGraph, hgu95av2.db, geneplotter, knitr, msigdbr, markdown, statmod
Description The EnrichmentBrowser package implements essential functionality for the enrichment analysis of gene expression data. The analysis combines the advantages of set-based and network-based enrichment analysis in order to derive high-confidence gene sets and biological pathways that are differentially regulated in the expression data under investigation. Besides, the package facilitates the visualization and exploration of such sets and pathways.
License Artistic-2.0
BugReports https://github.com/lgeistlinger/EnrichmentBrowser/issues
Encoding UTF-8
VignetteBuilder knitr
biocViews ImmunoOncology, Microarray, RNASeq, GeneExpression, DifferentialExpression, Pathways, GraphAndNetwork, Network, GeneSetEnrichment, NetworkEnrichment, Visualization, ReportWriting
RoxygenNote 7.2.3
git_url https://git.bioconductor.org/packages/EnrichmentBrowser
git_branch RELEASE_3_18
git_last_commit a2bb60b
git_last_commit_date 2023-10-24
combResults

Combining enrichment analysis results

Description

Different enrichment analysis methods usually result in different gene set rankings for the same dataset. This function allows to combine results from the different set-based and network-based enrichment analysis methods. This includes the computation of average gene set ranks across methods.
Usage

```r
combResults(
  res.list,
  rank.col = configEBrowser("PVAL.COL"),
  decreasing = FALSE,
  rank.fun = c("comp.ranks", "rel.ranks", "abs.ranks"),
  comb.fun = c("mean", "median", "min", "max", "sum")
)
```

Arguments

- `res.list`: A list of enrichment analysis result lists (as returned by the functions `sbea` and `nbea`).
- `rank.col`: Rank column. Column name of the enrichment analysis result table that should be used to rank the gene sets. Defaults to the gene set p-value column, i.e. gene sets are ranked according to gene set significance.
- `decreasing`: Logical. Should smaller (decreasing=FALSE, default) or larger (decreasing=TRUE) values in `rank.col` be ranked better? In case of gene set p-values the smaller the better, in case of gene set scores the larger the better.
- `rank.fun`: Ranking function. Used to rank gene sets according to the result table of individual enrichment methods (as returned from the `gsRanking` function). This is typically done according to gene set p-values, but can also take into account gene set scores/statistics, especially in case of gene sets with equal p-value. Can be either one of the predefined functions ('comp.ranks', 'rel.ranks', 'abs.ranks') or a user-defined function. Defaults to 'comp.ranks', i.e. competitive (percentile) ranks are computed by calculating for each gene set the percentage of gene sets with a p-value as small or smaller. Alternatively, 'rel.ranks', i.e. relative ranks are computed in 2 steps:
  1. Ranks are assigned according to distinct gene set p-value *categories*, i.e. gene sets with equal p-value obtain the *same* rank. Thus, the gene sets with lowest p-value obtain rank 1, and so on.
  2. As opposed to absolute ranks (rank.fun = 'abs.ranks'), which are returned from step 1, relative ranks are then computed by dividing the absolute rank by number of distinct p-value categories and multiplying with 100 (= percentile rank).
- `comb.fun`: Rank combination function. Used to combine gene set ranks across methods. Can be either one of the predefined functions (mean, median, max, min, sum) or a user-defined function. Defaults to 'sum', i.e. the rank sum across methods is computed.

Value

An enrichment analysis result list that can be detailedly explored by calling `eaBrowse` and from which a flat gene set ranking can be extracted by calling `gsRanking`.

Author(s)

Ludwig Geistlinger
See Also

sbea, nbea, eaBrowse

Examples

# (1) expression data:
# simulated expression values of 100 genes
# in two sample groups of 6 samples each
se <- makeExampleData(what="SE")
se <- deAna(se)

# (2) gene sets:
# draw 10 gene sets with 15-25 genes
gs <- makeExampleData(what="gs", gnames=names(se))

# (3) make artificial enrichment analysis results:
# 2 ea methods with 5 significantly enriched gene sets each
ora.res <- makeExampleData(what="ea.res", method="ora", se=se, gs=gs)
gsea.res <- makeExampleData(what="ea.res", method="gsea", se=se, gs=gs)

# (4) combining the results
res.list <- list(ora.res, gsea.res)
comb.res <- combResults(res.list)

# (5) result visualization and exploration
gsRanking(comb.res)

# user-defined ranking and combination functions
# (a) dummy ranking, give 1:nrow(res.tbl)
dummy.rank <- function(res.tbl) seq_len(nrow(res.tbl))

# (b) weighted average for combining ranks
wavg <- function(r) mean(c(1,2) * r)

comb.res <- combResults(res.list, rank.fun=dummy.rank, comb.fun=wavg)

compileGRN

Compilation of a gene regulatory network from pathway databases

Description

To perform network-based enrichment analysis a gene regulatory network (GRN) is required. There are well-studied processes and organisms for which comprehensive and well-annotated regulatory networks are available, e.g. the RegulonDB for E. coli and YeastRACT for S. cerevisiae. However, in many cases such a network is missing. A first simple workaround is to compile a network from regulations in pathway databases such as KEGG.
Usage

```r
compileGRN(
  org,
  db = "kegg",
  act.inh = TRUE,
  map2entrez = TRUE,
  keep.type = FALSE,
  kegg.native = FALSE
)
```

Arguments

- `org`: An organism in KEGG three letter code, e.g. ‘hsa’ for ‘Homo sapiens’. Alternatively, and mainly for backward compatibility, this can also be either a list of `KEGGPathway` objects or an absolute file path of a zip compressed archive of pathway xml files in KGML format.
- `db`: Pathway database. This should be one or more DBs out of 'kegg', 'reactome', 'pathbank', and 'wikipathways'. See `pathwayDatabases` for available DBs of the respective organism. Default is 'kegg'. Note: when dealing with non-model organisms, GRN compilation is currently only supported directly from KEGG (the argument `kegg.native` should accordingly be set to `TRUE`).
- `act.inh`: Should gene regulatory interactions be classified as activating (+) or inhibiting (-)? If TRUE, this will drop interactions for which such a classification cannot be made (e.g. binding events). Otherwise, all interactions found in the pathway DB will be included. Default is `TRUE`.
- `map2entrez`: Should gene identifiers be mapped to NCBI Entrez Gene IDs? This only applies to Reactome and PathBank as they both use UNIPROT IDs. This is typically recommended when using the GRN for network-based enrichment analysis with the EnrichmentBrowser. Default is `TRUE`.
- `keep.type`: Should the original interaction type descriptions be kept? If TRUE, this will keep the long description of interaction types as found in the original KGML and BioPax pathway files. Default is `FALSE`.
- `kegg.native`: For KEGG: should the GRN be compiled from the native KGML files or should graphite’s pathway topology conversion be used? See the vignette of the graphite package for details. This is mostly for backward compatibility. Default is `FALSE`. Note: when dealing with non-model organisms (not supported by graphite) this argument should be set to `TRUE`.

Value

The GRN in plain matrix format. Two columns named `FROM` (the regulator) and `TO` (the regulated gene) are guaranteed. Additional columns, named `TYPE` and `LONG.TYPE`, are included if option `act.inh` or `keep.type` is activated.

Author(s)

Ludwig Geistlinger
See Also

pathwayDatabases, pathways, KEGGPathway, parseKGML, downloadPathways

Examples

kegg.grn <- compileGRN(org="hsa", db="kegg")

---

configEBrowser \(\text{Configuring the EnrichmentBrowser}\)

Description

Function to get and set configuration parameters determining the default behavior of the EnrichmentBrowser

Usage

configEBrowser(key, value = NULL)

Arguments

key Configuration parameter.
value Value to overwrite the current value of key.

Details

Important colData, rowData, and result column names:

- SMPL.COL: colData column storing the sample IDs (default: "SAMPLE")
- GRP.COL: colData column storing binary group assignment (default: "GROUP")
- BLK.COL: colData column defining paired samples or sample blocks (default: "BLOCK")
- PRB.COL: rowData column storing probe/feature IDs ("PROBEID", read-only)
- EZ.COL: rowData column storing gene ENTREZ IDs ("ENTREZID", read-only)
- SYM.COL: rowData column storing gene symbols ("SYMBOL", read-only)
- GN.COL: rowData column storing gene names ("GENENAME", read-only)
- FC.COL: rowData column storing (log2) fold changes of differential expression between sample groups (default: "FC")
- ADJP.COL: rowData column storing adjusted (corrected for multiple testing) p-values of differential expression between sample groups (default: "ADJ.PVAL")
- GS.COL: result table column storing gene set IDs (default: "GENE.SET")
- PVAL.COL: result table column storing gene set significance (default: "PVAL")
- PMID.COL: gene table column storing PUBMED IDs ("PUBMED", read-only)
Important URLs (all read-only):

- KEGG.URL: http://www.genome.jp/dbget-bin/
- KEGG.GENE.URL: http://www.genome.jp/dbget-bin/www_bget?
- KEGG.SHOW.URL: http://www.genome.jp/dbget-bin/show_pathway?
- GO.SHOW.URL: http://amigo.geneontology.org/amigo/term/

Default output directory:

- EBROWSER.HOME: tools::R_user_dir("EnrichmentBrowser")
- OUTDIR.DEFAULT: file.path(EBROWSER.HOME, "results")

Gene set size:

- GS.MIN.SIZE: minimum number of genes per gene set (default: 5)
- GS.MAX.SIZE: maximum number of genes per gene set (default: 500)

Result appearance:

- RESULT.TITLE: (default: "Table of Results")
- NR.SHOW: maximum number of entries to show (default: 20)

Value

If is.null(value) this returns the value of the selected configuration parameter. Otherwise, it updates the selected parameter with the given value.

Author(s)

Ludwig Geistlinger

Examples

```r
# getting config information
cfgEBrowser("GS.MIN.SIZE")

# setting config information
# WARNING: this is for advanced users only!
# inappropriate settings will impair EnrichmentBrowser's functionality
cfgEBrowser(key="GS.MIN.SIZE", value=3)

# restoring default config settings
cfgEBrowser()
```
Description

The function carries out a differential expression analysis between two sample groups. Resulting fold changes and derived p-values are returned. Raw p-values are corrected for multiple testing.

Usage

deAna(
  expr,
  grp = NULL,
  blk = NULL,
  de.method = c("limma", "edgeR", "DESeq2"),
  padj.method = "BH",
  stat.only = FALSE,
  filter.by.expr = TRUE,
  assay = "auto"
)

Arguments

expr  Expression data. A numeric matrix. Rows correspond to genes, columns to samples. Alternatively, this can also be an object of class SummarizedExperiment.

grp   *BINARY* group assignment for the samples. Use '0' and '1' for unaffected (controls) and affected (cases) samples, respectively. If NULL, this is assumed to be defined via a column named 'GROUP' in the colData slot if 'expr' is a SummarizedExperiment.

blk   Optional. For paired samples or sample blocks. This can also be defined via a column named 'BLOCK' in the colData slot if 'expr' is a SummarizedExperiment.

de.method  Differential expression method. Use 'limma' for microarray and RNA-seq data. Alternatively, differential expression for RNA-seq data can also be calculated using edgeR ('edgeR') or DESeq2 ('DESeq2'). Defaults to 'limma'.

padj.method Method for adjusting p-values to multiple testing. For available methods see the man page of the stats function p.adjust. Defaults to 'BH'.

stat.only  Logical. Should only the test statistic be returned? This is mainly for internal use, in order to carry out permutation tests on the DE statistic for each gene. Defaults to FALSE.

filter.by.expr  Logical. For RNA-seq data: include only genes with sufficiently large counts in the DE analysis? If TRUE, excludes genes not satisfying a minimum number of read counts across samples using the filterByExpr function from the edgeR package. Defaults to TRUE.
assay

Character. The name of the assay for differential expression analysis if expr is a SummarizedExperiment with multiple assays. Defaults to "auto", which automatically determines the appropriate assay based on data type provided and DE method selected. See details.

Details

Using a SummarizedExperiment with multiple assays:

For the typical use case within the EnrichmentBrowser workflow this will be a SummarizedExperiment with two assays: (i) an assay storing the raw expression values, and (ii) an assay storing the normalized expression values as obtained with the normalize function.

In this case, assay = "auto" will automatically determine the assay based on the data type provided. For microarray data, differential expression analysis will be carried out on the assay storing the normalized log2 intensities. For RNA-seq data, differential expression analysis will be carried out on the assay storing the raw read counts.

For usage outside of the typical workflow, the assay argument can be used to provide the name of the assay for differential expression analysis. For differential expression analysis of microarray data with de.method = "limma", this assay should contain the normalized log2 intensities. For differential expression analysis of RNA-seq data with either method (limma/voom, edgeR, or DESeq2), the specified assay should contain the raw read counts.

Value

A DE-table with measures of differential expression for each gene/row, i.e. a two-column matrix with log2 fold changes in the 1st column and derived p-values in the 2nd column. If 'expr' is a SummarizedExperiment, the DE-table will be automatically appended to the rowData slot.

Author(s)

Ludwig Geistlinger

See Also

readSE for reading expression data from file, normalize for normalization of expression data, voom for preprocessing of RNA-seq data, p.adjust for multiple testing correction, eBayes for DE analysis with limma, glmQLFit for DE analysis with edgeR, and DESeq for DE analysis with DESeq2.

Examples

# (1) microarray data: intensity measurements
maSE <- makeExampleData(what = "SE", type = "ma")
maSE <- deAna(maSE)
rowData(maSE)

# (2) RNA-seq data: read counts
rseqSE <- makeExampleData(what = "SE", type = "rseq")
rseqSE <- deAna(rseqSE, de.method = "DESeq2")
rowData(rseqSE)
downloadPathways  Download of KEGG pathways for a particular organism

Description

The function downloads all metabolic and non-metabolic pathways in KEGG XML format for a specified organism.

Usage

downloadPathways(org, cache = TRUE, out.dir = NULL, zip = FALSE)

Arguments

org  Organism in KEGG three letter code, e.g. 'hsa' for 'homo sapiens'.
cache Logical. Should a locally cached version used if available? Defaults to TRUE.
out.dir  Output directory. If not null, pathways are written to files in the specified directory.
zip Logical. In case pathways are written to file (‘out.dir’ is not null): should output files be zipped?

Value

if(is.null(out.dir)): a list of KEGGPathway objects else: none, as pathways are written to file

Author(s)

Ludwig Geistlinger

See Also

keggList, keggGet, KEGGPathway, parseKGML

Examples

pwys <- downloadPathways("hsa")
Exploration of enrichment analysis results

Description

Functions to extract a flat gene set ranking from an enrichment analysis result object and to de-
tailiedly explore it.

Usage

eaBrowse(
    res,
    nr.show = -1,
    graph.view = NULL,
    html.only = FALSE,
    out.dir = NULL,
    report.name = NULL
)

gsRanking(res, signif.only = TRUE)

Arguments

res      Enrichment analysis result list (as returned by the functions sbea and nbea).
nr.show  Number of gene sets to show. As default all statistically significant gene sets are
displayed.
graph.view Optional. Should a graph-based summary (reports and visualizes consistency
of regulations) be created for the result? If specified, it needs to be a gene
regulatory network, i.e. either an absolute file path to a tabular file or a character
matrix with exactly *THREE* cols; 1st col = IDs of regulating genes; 2nd col
= corresponding regulated genes; 3rd col = regulation effect; Use ’+’ and ’-’ for
activation/inhibition.
html.only Logical. Should the html file only be written (without opening the browser to
view the result page)? Defaults to FALSE.
out.dir   Output directory. If NULL, defaults to a timestamp-generated subdirectory of
configEBrowser("OUTDIR.DEFAULT").
report.name Name of the HTML report. If NULL, defaults to the enrichment method used.
signif.only Logical. Display only those gene sets in the ranking, which satisfy the signifi-
cance level? Defaults to TRUE.

Value

gsRanking: DataFrame with gene sets ranked by the corresponding p-value;
eaBrowse: none, opens the browser to explore results.
If not instructed otherwise (via argument out.dir), the main HTML report and associated files are written to configEBrowser("OUTDIR.DEFAULT"). See `?configEBrowser` to change the location. If html.only=FALSE, the HTML report will automatically be opened in your default browser.

Author(s)

Ludwig Geistlinger

See Also

sbea, nbea, combResults

Examples

```r
# real data
dir <- system.file("extdata", package="EnrichmentBrowser")
exprs.file <- file.path(dir, "exprs.tab")
cdat.file <- file.path(dir, "colData.tab")
rdat.file <- file.path(dir, "rowData.tab")
probeSE <- readSE(exprs.file, cdat.file, rdat.file)
geneSE <- probe2gene(probeSE)
geneSE <- deAna(geneSE)
metadata(geneSE)$annotation <- "hsa"

# artificial enrichment analysis results
gs <- makeExampleData(what="gs", gnames=names(geneSE))
ea.res <- makeExampleData(what="ea.res", method="ora", se=geneSE, gs=gs)

dir <- configEBrowser("OUTDIR.DEFAULT")
eaBrowse(ea.res, out.dir=dir, report.name="oraReport")
```

Description

This is the all-in-one wrapper function to perform the standard enrichment analysis pipeline implemented in the EnrichmentBrowser package.

Usage

```r
ebrowser(
  meth, 
  exprs, 
  cdat, 
  ... 
)```
ebrowser

```r
rdat,
org,
data.type = c(NA, "ma", "rseq"),
norm.method = "quantile",
dc.method = "limma",
gs,
grn = NULL,
perm = 1000,
alpha = 0.05,
beta = 1,
comb = FALSE,
browse = TRUE,
nr.show = -1,
out.dir = NULL,
report.name = "index",
...
)
```

Arguments

meth
Enrichment analysis method(s). See `sbeaMethods` and `nbeaMethods` for currently supported enrichment analysis methods. See also `sbea` and `nbea` for details.

exprs
Expression matrix. A tab separated text file containing the expression values (microarray: intensity measurements, RNA-seq: read counts). Columns = samples/subjects; rows = features/probes/genes; NO headers, row or column names. Alternatively, this can be a `SummarizedExperiment`, assuming the expression matrix in the `assays` slot. See details.

cdat
Column (phenotype) data. A tab separated text file containing annotation information for the samples in either *two or three* columns. NO headers, row or column names. The number of rows/samples in this file should match the number of columns/samples of the expression matrix. The 1st column is reserved for the sample IDs; The 2nd column is reserved for a *BINARY* group assignment. Use '0' and '1' for unaffected (controls) and affected (cases) sample class, respectively. For paired samples or sample blocks a third column is expected that defines the blocks. If `exprs` is a `SummarizedExperiment`, the `cdat` argument can be left unspecified, which then expects group and optional block assignments in respectively named columns 'GROUP' (mandatory) and 'BLOCK' (optional) in the `colData` slot.

rdat
Row (feature) data. A tab separated text file containing annotation information for the features. In case of probe level data: exactly *TWO* columns; 1st col = probe/feature IDs; 2nd col = corresponding gene ID for each feature ID in 1st col. In case of gene level data: the gene IDs newline-separated (i.e. just *one* column). It is recommended to use *ENTREZ* gene IDs (to benefit from downstream visualization and exploration functionality of the EnrichmentBrowser). NO headers, row or column names. The number of rows (features/probes/genes) in this file should match the number of rows/features of the expression matrix. Alternatively, this can also be the ID of a recognized platform such as
'hgu95av2' (Affymetrix Human Genome U95 chip) or 'ecoli2' (Affymetrix E. coli Genome 2.0 Array). If 'exprs' is a \texttt{SummarizedExperiment}, the \texttt{rdat} argument can be left unspecified, which then expects probe and corresponding Entrez Gene IDs in respectively named columns `PROBEID` and `ENTREZID` in the \texttt{rowData} slot.

\textbf{org} \hspace{1cm} Organism under investigation in KEGG three letter code, e.g. `hsa` for `Homo sapiens`. See also \texttt{kegg.species.code} to convert your organism of choice to KEGG three letter code.

\textbf{data.type} \hspace{1cm} Expression data type. Use `ma` for microarray and `rseq` for RNA-seq data. If NA, data.type is automatically guessed. If the expression values in `exprs` are decimal numbers they are assumed to be microarray intensities. Whole numbers are assumed to be RNA-seq read counts. Defaults to NA.

\textbf{norm.method} \hspace{1cm} Determines whether and how the expression data should be normalized. For available microarray normalization methods see the man page of the \texttt{limma} function \texttt{normalizeBetweenArrays}. For available RNA-seq normalization methods see the man page of the \texttt{EDASeq} function \texttt{betweenLaneNormalization}. Defaults to `quantile`, i.e. normalization is carried out so that quantiles between arrays/lanes/samples are equal. Use `none` to indicate that the data is already normalized and should not be normalized by ebrowser. See the man page of \texttt{normalize} for details.

\textbf{de.method} \hspace{1cm} Determines which method is used for per-gene differential expression analysis. See the man page of \texttt{deAna} for details. Defaults to `limma`, i.e. differential expression is calculated based on the typical \texttt{limma} \texttt{lmFit} procedure. This can also be `none` to indicate that DE analysis has already been carried out and should not be overwritten by ebrowser (applies only when \texttt{exprs} is given as a \texttt{SummarizedExperiment}).

\textbf{gs} \hspace{1cm} Gene sets. Either a list of gene sets (character vectors of gene IDs) or a text file in GMT format storing all gene sets under investigation.

\textbf{grn} \hspace{1cm} Gene regulatory network. Either an absolute file path to a tabular file or a character matrix with exactly *THREE* cols; 1st col = IDs of regulating genes; 2nd col = corresponding regulated genes; 3rd col = regulation effect; Use `+` and `-` for activation/inhibition.

\textbf{perm} \hspace{1cm} Number of permutations of the sample group assignments. Defaults to 1000. Can also be an integer vector matching the length of `meth` to assign different numbers of permutations for different methods.

\textbf{alpha} \hspace{1cm} Statistical significance level. Defaults to 0.05.

\textbf{beta} \hspace{1cm} Log2 fold change significance level. Defaults to 1 (2-fold).

\textbf{comb} \hspace{1cm} Logical. Should results be combined if more then one enrichment method is selected? Defaults to FALSE.

\textbf{browse} \hspace{1cm} Logical. Should results be displayed in the browser for interactive exploration? Defaults to TRUE.

\textbf{nr.show} \hspace{1cm} Number of gene sets to show. As default all statistical significant gene sets are displayed. Note that this only influences the number of gene sets for which additional visualization will be provided (typically only of interest for the top / significant gene sets). Selected enrichment methods and resulting flat gene set rankings still include the complete number of gene sets under study.
Details

Given flat gene expression data, the data is read in and subsequently subjected to chosen enrichment analysis methods.

The results from different methods can be combined and investigated in detail in the default browser.

*On data type and normalization:*

Normalization of high-throughput expression data is essential to make results within and between experiments comparable. Microarray (intensity measurements) and RNA-seq (read counts) data exhibit typically distinct features that need to be normalized for. This function wraps commonly used functionality from limma for microarray normalization and from EDASeq for RNA-seq normalization. For specific needs that deviate from standard normalizations, the user should always refer to more specific functions/packages. See also the limma’s user guide http://www.bioconductor.org/packages/limma for definition and normalization of the different expression data types.

Microarray data is expected to be single-channel. For two-color arrays, it is expected here that normalization within arrays has been already carried out, e.g. using normalizeWithinArrays from limma.

RNA-seq data is expected to be raw read counts. Please note that normalization for downstream DE analysis, e.g. with edgeR and DESeq2, is not ultimately necessary (and in some cases even discouraged) as many of these tools implement specific normalization approaches. See the vignette of EDASeq, edgeR, and DESeq2 for details.

Value

None, writes an HTML report and, if selected, opens the browser to explore results. If not instructed otherwise (via argument out.dir), the main HTML report and associated files are written to configEBrowser("OUTDIR.DEFAULT"). See ?configEBrowser to change the location. If browse=TRUE, the HTML report will automatically be opened in the default browser.

Author(s)

Ludwig Geistlinger

References

Limma User’s guide: http://www.bioconductor.org/packages/limma

See Also

readSE to read expression data from file; probe2gene to transform probe to gene level expression; kegg.species.code maps species name to KEGG code. getGenesets to retrieve gene set databases such as GO or KEGG; compileGRN to construct a GRN from pathway databases; sbea to perform set-based enrichment analysis; nbea to perform network-based enrichment analysis;
**combResults** to combine results from different methods; **eaBrowse** for exploration of resulting gene sets

### Examples

```r
# expression data from file
eprs.file <- system.file("extdata/exprs.tab", package="EnrichmentBrowser")
cdat.file <- system.file("extdata/colData.tab", package="EnrichmentBrowser")
rdat.file <- system.file("extdata/rowData.tab", package="EnrichmentBrowser")

# getting all human KEGG gene sets
# hsa.gs <- getGenesets(org="hsa", db="kegg")
gs.file <- system.file("extdata/hsa_kegg_gs.gmt", package="EnrichmentBrowser")
hsa.gs <- getGenesets(gs.file)

# output destination
out.dir <- configEBrowser("OUTDIR.DEFAULT")

# set-based enrichment analysis
ebrowser( meth="ora", perm=0,
    exprs=exprs.file, cdat=cdat.file, rdat=rdat.file,
    gs=hsa.gs, org="hsa", nr.show=3,
    out.dir=out.dir, report.name="oraReport")

# compile a gene regulatory network from KEGG pathways
hsa.grn <- compileGRN(org="hsa", db="kegg")

# network-based enrichment analysis
ebrowser( meth="ggea",
    exprs=exprs.file, cdat=cdat.file, rdat=rdat.file,
    gs=hsa.gs, grn=hsa.grn, org="hsa", nr.show=3,
    out.dir=out.dir, report.name="ggeaReport")

# combining results
ebrowser( meth=c("ora", "ggea"), perm=0, comb=TRUE,
    exprs=exprs.file, cdat=cdat.file, rdat=rdat.file,
    gs=hsa.gs, grn=hsa.grn, org="hsa", nr.show=3,
    out.dir=out.dir, report.name="combReport")
```

---

**Export results from set- and network-based enrichment analysis**

### Description

This function exports results of differential expression (DE) analysis such as enrichplot and GOPlot.

### Usage

```r
export(res, to = c("enrichplot", "GOPlot"))
```
getGenesets

Arguments

res

Enrichment analysis results as returned by sbea or nbea.

to

Character. Downstream package to which export enrichment analysis results to. Defaults to "enrichplot", currently the only supported export option.

Description

Functionality for retrieving gene sets for an organism under investigation from databases such as GO and KEGG. Parsing and writing a list of gene sets from/to a flat text file in GMT format is also supported.

The GMT (Gene Matrix Transposed) file format is a tab delimited file format that describes gene sets. In the GMT format, each row represents a gene set. Each gene set is described by a name, a description, and the genes in the gene set. See references.

Usage

getGenesets(
  org,
  db = c("go", "kegg", "msigdb", "enrichr"),
  gene.id.type = "ENTREZID",
  cache = TRUE,
  return.type = c("list", "GeneSetCollection"),
  ...
)

showAvailableSpecies(db = c("go", "kegg", "msigdb", "enrichr"), cache = TRUE)

showAvailableCollections(
  org,
  db = c("go", "kegg", "msigdb", "enrichr"),
  cache = TRUE
)

writeGMT(gs, gmt.file)

Arguments

org

An organism in (KEGG) three letter code, e.g. ‘hsa’ for ‘Homo sapiens’. Alternatively, this can also be a text file storing gene sets in GMT format. See details.

db

Database from which gene sets should be retrieved. Currently, either ‘go’ (default), ‘kegg’, ‘msigdb’, or ‘enrichr’.
getGenesets

gene.id.type  Character. Gene ID type of the returned gene sets. Defaults to "ENTREZID". See idTypes for available gene ID types.
cache  Logical. Should a locally cached version used if available? Defaults to TRUE.
return.type  Character. Determines whether gene sets are returned as a simple list of gene sets (each being a character vector of gene IDs), or as an object of class GeneSetCollection.

...  Additional arguments for individual gene set databases. For db = "GO":

• onto: Character. Specifies one of the three GO ontologies: 'BP' (biological process), 'MF' (molecular function), 'CC' (cellular component). Defaults to 'BP'.
• evid: Character. Specifies one or more GO evidence code(s) such as IEP (inferred from expression pattern) or TAS (traceable author statement). Defaults to NULL which includes all annotations, i.e. does not filter by evidence codes. See references for a list of available evidence codes.
• hierarchical: Logical. Incorporate hierarchical relationships between GO terms ('is_a' and 'has_a') when collecting genes annotated to a GO term? If set to TRUE, this will return all genes annotated to a GO term *or to one of its child terms* in the GO ontology. Defaults to FALSE, which will then only collect genes directly annotated to a GO term.
• mode: Character. Determines in which way the gene sets are retrieved. This can be either 'GO.db' or 'biomart'. The 'GO.db' mode creates the gene sets based on BioC annotation packages - which is fast, but represents not necessarily the most up-to-date mapping. In addition, this option is only available for the currently supported model organisms in BioC. The 'biomart' mode downloads the mapping from BioMart - which can be time consuming, but allows to select from a larger range of organisms and contains the latest mappings. Defaults to 'GO.db'.

For db = "msigdb":

• cat: Character. MSigDB collection category: 'H' (hallmark), 'C1' (genomic position), 'C2' (curated databases), 'C3' (binding site motifs), 'C4' (computational cancer), 'C5' (Gene Ontology), 'C6' (oncogenic), 'C7' (immunologic), 'C8' (cell type). See references.
• subcat: Character. MSigDB collection subcategory. Depends on the chosen MSigDB collection category. For example, 'MIR' to obtain microRNA targets from the 'C3' collection. See references.

For db = "enrichr":

• lib: Character. Enrichr gene set library. For example, 'Genes_Associated_with_NIH_Grants' to obtain gene sets based on associations with NIH grants. See references.

gs  A list of gene sets (character vectors of gene IDs).
gmt.file  Gene set file in GMT format. See details.

Value

For getGenesets: a list of gene sets (vectors of gene IDs). For writeGMT: none, writes to file.

For showAvailableSpecies and showAvailableCollections: a DataFrame, displaying supported species and available gene set collections for a gene set database of choice.
getGenesets

Author(s)

Ludwig Geistlinger

References

GO: http://geneontology.org/
GO evidence codes: http://geneontology.org/docs/guide-go-evidence-codes/
KEGG Organism code: http://www.genome.jp/kegg/catalog/org_list.html
MSigDB: http://software.broadinstitute.org/gsea/msigdb/collections.jsp
Enrichr: https://maayanlab.cloud/Enrichr/#stats

See Also

the GO.db package for GO2gene mapping used in 'GO.db' mode, and the biomaRt package for general queries to BioMart.
keggList and keggLink for accessing the KEGG REST server.
msigdb::msigdbr for obtaining gene sets from the MSigDB.

Examples

# (1) Typical usage for gene set enrichment analysis with GO:
# Biological process terms based on BioC annotation (for human)
go.gs <- getGenesets(org = "hsa", db = "go")

# eq.:  
# go.gs <- getGenesets(org = "hsa", db = "go", onto = "BP", mode = "GO.db")

# Alternatively:
# downloading from BioMart
# this may take a few minutes ...
go.gs <- getGenesets(org = "hsa", db = "go", mode = "biomart")

# list supported species for obtaining gene sets from GO
showAvailableSpecies(db = "go")

# (2) Defining gene sets according to KEGG
kegg.gs <- getGenesets(org = "hsa", db = "kegg")

# list supported species for obtaining gene sets from KEGG
showAvailableSpecies(db = "kegg")

# (3) Obtaining *H*allmark gene sets from MSigDB
hall.gs <- getGenesets(org = "hsa", db = "msigdb", cat = "H")

# list supported species for obtaining gene sets from MSigDB
showAvailableSpecies(db = "msigdb")
GGEA graphs of consistency between regulation and expression

Description

Gene graph enrichment analysis (GGEA) is a network-based enrichment analysis method implemented in the EnrichmentBrowser package. The idea of GGEA is to evaluate the consistency of known regulatory interactions with the observed gene expression data. A GGEA graph for a gene set of interest displays the consistency of each interaction in the network that involves a gene set member. Nodes (genes) are colored according to expression (up-/down-regulated) and edges (interactions) are colored according to consistency, i.e. how well the interaction type (activation/inhibition) is reflected in the correlation of the expression of both interaction partners.

Usage

ggeaGraph(
  gs,
  grn,
  se,
  alpha = 0.05,
  beta = 1,
  max.edges = 50,
  cons.thresh = 0.7,
  show.scores = FALSE
)

ggeaGraphLegend()
Arguments

- `gs` Gene set under investigation. This should be a character vector of gene IDs.
- `grn` Gene regulatory network. Character matrix with exactly *THREE* cols; 1st col = IDs of regulating genes; 2nd col = corresponding regulated genes; 3rd col = regulation effect; Use `'+'` and `'-'` for activation/inhibition.
- `se` Expression data given as an object of class `SummarizedExperiment`.
- `alpha` Statistical significance level. Defaults to 0.05.
- `beta` Log2 fold change significance level. Defaults to 1 (2-fold).
- `max.edges` Maximum number of edges that should be displayed. Defaults to 50.
- `cons.thresh` Consistency threshold. Graphical parameter that correspondingly increases line width of edges with a consistency above the chosen threshold (defaults to 0.7).
- `show.scores` Logical. Should consistency scores of the edges be displayed? Defaults to FALSE.

Value
None, plots to a graphics device.

Author(s)
Ludwig Geistlinger

See Also
- `nbea` to perform network-based enrichment analysis. `eaBrowse` for exploration of resulting gene sets.

Examples

```r
# (1) expression data:
# simulated expression values of 100 genes
# in two sample groups of 6 samples each
se <- makeExampleData(what="SE")
se <- deAna(se)

# (2) gene sets:
# draw 10 gene sets with 15-25 genes
gs <- makeExampleData(what="gs", gnames=names(se))

# (3) compiling artificial regulatory network
grn <- makeExampleData(what="grn", nodes=names(se))

# (4) plot consistency graph
ggeaGraph(gs=gs[[1]], grn=grn, se=se)

# (5) get legend
ggeaGraphLegend()
```
idMap

Mapping between gene ID types

Description

Functionality to map between common gene ID types such as ENSEMBL and ENTREZ for gene expression datasets, gene sets, and gene regulatory networks.

Usage

```r
idMap(
  obj,
  org = NA,
  from = "ENSEMBL",
  to = "ENTREZID",
  multi.to = "first",
  multi.from = "first"
)

idTypes(org)
```

Arguments

- **obj**: The object for which gene IDs should be mapped. Supported options include
  - Gene expression dataset. An object of class `SummarizedExperiment`. Expects the names to be of gene ID type given in argument `from`.
  - Gene sets. Either a list of gene sets (character vectors of gene IDs) or a `GeneSetCollection` storing all gene sets.
  - Gene regulatory network. A 3-column character matrix; 1st col = IDs of regulating genes; 2nd col = IDs of regulated genes; 3rd col = regulation effect; Use `'+`' and `-` for activation/inhibition.
- **org**: Character. Organism in KEGG three letter code, e.g. 'hsa' for 'Homo sapiens'. See references.
- **from**: Character. Gene ID type from which should be mapped. Corresponds to the gene ID type of argument `obj`. Defaults to ENSEMBL.
- **to**: Character. Gene ID type to which should be mapped. Corresponds to the gene ID type the argument `obj` should be updated with. If `obj` is an expression dataset of class `SummarizedExperiment`, to can also be the name of a column in the `rowData` slot to specify user-defined mappings in which conflicts have been manually resolved. Defaults to ENTREZID.
- **multi.to**: How to resolve 1:many mappings, i.e. multiple to.IDs for a single from.ID? This is passed on to the `multiVals` argument of `mapIds` and can thus take several pre-defined values, but also the form of a user-defined function. However, note that this requires that a single to.ID is returned for each from.ID. Default is "first", which accordingly returns the first to.ID mapped onto the respective from.ID.
multi.from

How to resolve many:1 mappings, i.e. multiple from.IDs mapping to the same to.ID? Only applicable if obj is an expression dataset of class `SummarizedExperiment`. Pre-defined options include:

- 'first' (Default): returns the first from.ID for each to.ID with multiple from.IDs,
- 'minp': selects the from.ID with minimum p-value (according to the rowData column `PVAL` of obj),
- 'maxfc': selects the from.ID with maximum absolute log2 fold change (according to the rowData column `FC` of obj).

Note that a user-defined function can also be supplied for custom behaviors. This will be applied for each case where there are multiple from.IDs for a single to.ID, and accordingly takes the arguments ids and obj. The argument ids corresponds to the multiple from.IDs from which a single ID should be chosen, e.g. via information available in argument obj. See examples for a case where ids are selected based on a user-defined rowData column.

Details

The function 'idTypes' lists the valid values which the arguments 'from' and 'to' can take. This corresponds to the names of the available gene ID types for the mapping.

Value

idTypes: character vector listing the available gene ID types for the mapping.

idMap: An object of the same class as the input argument obj, i.e. a `SummarizedExperiment` if provided an expression dataset, a list of character vectors or a `GeneSetCollection` if provided gene sets, and a character matrix if provided a gene regulatory network.

Author(s)

Ludwig Geistlinger

References


See Also

`SummarizedExperiment`, `mapIds`, `keytypes`

Examples

```r
# (1) ID mapping for gene expression datasets
# create an expression dataset with 3 genes and 3 samples
se <- makeExampleData("SE", nfeat = 3, nsmpl = 3)
names(se) <- paste0("ENSG00000000", c("003", "005", "419"))
idMap(se, org = "hsa")

# user-defined mapping
rowData(se)$MYID <- c("g1", "g1", "g2")
```
import idMap(se, to = "MYID")

# data-driven resolving of many:1 mappings

## e.g. select from.ID with lowest p-value
pcol <- configEBrowser("PVAL.COL")
rowData(se)[[pcol]] <- c(0.001, 0.32, 0.15)
idMap(se, to = "MYID", multi.from = "minp")

## ... or using a customized function
maxScore <- function(ids, se)
{
    scores <- rowData(se)[ids, "SCORE"]
    ind <- which.max(scores)
    return(ids[ind])
}
rowData(se)$SCORE <- c(125.7, 33.4, 58.6)
idMap(se, to = "MYID", multi.from = maxScore)

# (2) ID mapping for gene sets
# create two gene sets containing 3 genes each
s2 <- paste0("ENSG00000", c("012048", "139618", "141510"))
gs <- list(s1 = names(se), s2 = s2)
idMap(gs, org = "hsa", from = "ENSEMBL", to = "SYMBOL")

# (3) ID mapping for gene regulatory networks
grn <- cbind(FROM = gs$s1, TO = gs$s2, TYPE = rep("+", 3))
idMap(grn, org = "hsa", from = "ENSEMBL", to = "ENTREZID")

---

### import

**Import results from differential expression (DE) analysis**

#### Description

This function imports fully processed expression datasets and results of differential expression (DE) analysis from limma, edgeR, and DESeq2. The imported data is converted to a `SummarizedExperiment`, annotating experimental design and genewise differential expression, which then allows straightforward application of enrichment analysis methods.

#### Usage

```r
import(obj, res, from = c("auto", "limma", "edgeR", "DESeq2"), anno = NA)
```

#### Arguments

- **obj**
  - Expression data object. Supported options include `EList` (voom/limma), `DGEList` (edgeR), and `DESeqDataSet` (DESeq2). See details.

- **res**
  - Differential expression results. Expected to match the provided expression data object type, i.e. should be an object of class
import

- `data.frame` if `obj` is provided as an `EList` (voom/limma),
- `TopTags` if `obj` is provided as a `DGEList` (edgeR), and
- `DESeqResults` if `obj` is provided as a `DESeqDataSet` (DESeq2). See details.

from Character. Differential expression method from which to import results from. Defaults to "auto", which automatically determines the import type based on the expression data object provided. Can also be explicitly chosen as either 'limma', 'edgeR' or 'DESeq2'.

anno Character. The organism under study in KEGG three letter code (e.g. 'hsa' for 'Homo sapiens'). For microarray probe-level data: the ID of a recognized microarray platform. See references.

Details

The expression data object (argument `obj`) is expected to be fully processed (including normalization and dispersion estimation) and to have the experimental design annotated. The experimental design is expected to describe *a comparison of two groups* with an optional blocking variable for paired samples / sample batches (i.e. `design = ~ group` or `design = ~ batch + group`).

The differential expression result (argument `res`) is expected to have the same number of rows as the expression data object, and also that the order of the rows is the same / consistent, i.e. that there is a 1:1 correspondence between the rownames of `obj` and the rownames of `res`. Note that the expression dataset is automatically restricted to the genes for which DE results are available. However, for an appropriate estimation of the size of the universe for competitive gene set tests, it is recommended to provide DE results for all genes in the expression data object whenever possible (see examples).

Value

An object of class `SummarizedExperiment` that stores

- the expression matrix in the `assay` slot,
- information about the samples, including the experimental design, in the `colData` slot, and
- information about the genes, including measures of differential expression, in the `rowData` slot.

Mandatory annotations:

- `colData` column storing binary group assignment (named "GROUP")
- `rowData` column storing (log2) fold changes of differential expression between sample groups (named "FC")
- `rowData` column storing adjusted (corrected for multiple testing) p-values of differential expression between sample groups (named "ADJ.PVAL").

Additional optional annotations:

- `colData` column defining paired samples or sample blocks (named "BLOCK")
- metadata slot named "annotation" giving the organism under investigation in KEGG three letter code (e.g. "hsa" for Homo sapiens)
- metadata slot named "dataType" indicating the expression data type ("ma" for microarray, "rseq" for RNA-seq).
import

Author(s)
Ludwig Geistlinger

References
KEGG Organism code http://www.genome.jp/kegg/catalog/org_list.html
Recognized microarray platforms http://www.bioconductor.org/packages/release/BiocViews.html#___ChipName

See Also
readSE for reading expression data from file, normalize for normalization of expression data, voom for preprocessing of RNA-seq data, p.adjust for multiple testing correction, eBayes for DE analysis with limma, glmQLFit for DE analysis with edgeR, and DESeq for DE analysis with DESeq2.

Examples

# Setup
## i) Generate count data
nsamples <- 4
genesis <- 1000
dispers <- 1 / rchisq(ngenes, df = 10)
rdesign <- model.matrix(~factor(rep(c(1, 2), each = 2)))
counts <- rnbinom(ngenes * nsamples, mu = 20, size = 1 / dispers)
counts <- matrix(counts, nrow = ngenes, ncol = nsamples)

## ii) Generate intensity data
sd <- 0.3 * sqrt(4 / rchisq(100, df = 4))
intens <- matrix(rnorm(100 * 6, sd = sd), nrow = 100, ncol = 6)
rownames(intens) <- paste0("Gene", 1:100)
mdesign <- cbind(Grp1 = 1, Grp2vs1 = rep(c(0,1), each = 3))

# (1) import from edgeR (RNA-seq count data)
# (1a) create the expression data object
library(edgeR)
d <- DGEList(counts)
d <- calcNormFactors(d)
d <- estimateDisp(d, rdesign)
# (1b) obtain differential expression results
fit <- glmQLFit(d, rdesign)
qlf <- glmQLFTest(fit)
res <- topTags(qlf, n = nrow(d), sort.by = "none")

# (1c) import
se <- import(d, res)

# (2) import from DESeq2 (RNA-seq count data)
# (2a) create the expression data object
library(DESeq2)
condition <- factor(rep(c("A", "B"), each = 2))
dds <- DESeqDataSetFromMatrix(counts,  
    colData = DataFrame(condition = condition),  
    design = ~ condition)

# (2b) obtain differential expression results
dds <- DESeq(dds)
res <- results(dds)

# (2c) import
se <- import(dds, res)

# (3) import from voom/limma (RNA-seq count data)
# (3a) create the expression data object
library(limma)
keep <- filterByExpr(counts, rdesign)
el <- voom(counts[keep,], rdesign)

# (3b) obtain differential expression results
fit <- lmFit(el, rdesign)
fit <- eBayes(fit, robust = TRUE)
res <- topTable(fit, coef = 2, number = nrow(counts), sort.by = "none")

# (3c) import
se <- import(el, res)

# (4) import from limma-trend (RNA-seq count data)
# (4a) create the expression data object
logCPM <- edgeR::cpm(counts[keep,], log = TRUE, prior.count = 3)
el <- new("EList", list(E = logCPM, design = rdesign))

# (4b) obtain differential expression results
fit <- lmFit(el, rdesign)
fit <- eBayes(fit, trend = TRUE)
res <- topTable(fit, coef = 2, number = nrow(el), sort.by = "none")

# (4c) import
se <- import(el, res)

# (5) import from limma (microarray intensity data)
# (5a) create the expression data object
el <- new("EList", list(E = intens, design = mdesign))

# (5b) obtain differential expression results
fit <- lmFit(el, mdesign)
fit <- eBayes(fit, robust = TRUE)
res <- topTable(fit, coef = 2, number = nrow(el), sort.by = "none")

# (5c) import
se <- import(el, res)
isAvailable

Is a required package available?

Description

Convenience function for checking and installing required packages.

Usage

isAvailable(pkg, type = c("annotation", "software", "data"))

Arguments

pkg  Character vector of length 1. A valid name of an existing R package.

type  Character vector of length 1. What type of package is this? Choose one out of 'annotation', 'software', or 'data' package.

Details

Checks whether a required package is available in the library. If yes, the package is loaded via requireNamespace. If not, the package is optionally installed via install and, subsequently, loaded via requireNamespace.

Value

None. See details.

Author(s)

Ludwig Geistlinger <Ludwig.Giestlinger@sph.cuny.edu>

See Also

require, install

Examples

isAvailable("EnrichmentBrowser", type="software")
makeExampleData

Example data for the EnrichmentBrowser package

Description

Functionality to construct example data sets for demonstration. This includes expression data, gene sets, gene regulatory networks, and enrichment analysis results.

Usage

makeExampleData(what = c("SE", "gs", "grn", "ea.res"), ...)

Arguments

what

Kind of example data set to be constructed. This should be one out of:

• SE: SummarizedExperiment
• gs: Gene set list
• grn: Gene regulatory network
• ea.res: Enrichment analysis result object as returned by the functions sbea and nbea

... Additional arguments to fine-tune the specific example data sets.

For what='SE':

• type: Expression data type. Should be either 'ma' (Microarray intensity measurements) or 'rseq' (RNA-seq read counts).
• nfeat: Number of features/genes. Defaults to 100.
• nsmpl: Number of samples. Defaults to 12.
• blk: Create sample blocks. Defaults to TRUE.
• norm: Should the expression data be normalized? Defaults to FALSE.
• deAna: Should an differential expression analysis be carried out automatically? Defaults to FALSE.

For what='gs':

• gnames: gene names from which the sets will be sampled. Per default the sets will be drawn from c(g1, ..., g100).
• n: number of sets. Defaults to 10.
• min.size: minimal set size. Defaults to 15.
• max.size: maximal set size. Defaults to 25.

For what='grn':

• nodes: gene node names for which edges will be drawn. Per default node names will be c(g1, ..., g100).
• edge.node.ratio: ratio number of edges / number of nodes. Defaults to 3, i.e. creates 3 times more edges than nodes.

For what='ea.res':


nbeaMethods

Network-based enrichment analysis (NBEA)

Description

This is the main function for network-based enrichment analysis. It implements and wraps existing implementations of several frequently used methods and allows a flexible inspection of resulting gene set rankings.

Usage

nbeaMethods()

nbea(
    method = EnrichmentBrowser::nbeaMethods(),
    se,
    gs,
    grn,
    prune.grn = TRUE,
    alpha = 0.05,
    perm = 1000,
    padj.method = "none",
    out.file = NULL,
    browse = FALSE,
    assay = "auto",
    ...
)
Arguments

method  Network-based enrichment analysis method. Currently, the following network-based enrichment analysis methods are supported: 'ggea', 'spia', 'pathnet', 'degraph', 'topologygsa', 'ganpa', 'cepa', 'netgsa', and 'neat'. Default is 'ggea'. This can also be the name of a user-defined function implementing a method for network-based enrichment analysis. See Details.

se  Expression dataset. An object of class `SummarizedExperiment`. Mandatory minimal annotations:
  • colData column storing binary group assignment (named "GROUP")
  • rowData column storing (log2) fold changes of differential expression between sample groups (named "FC")
  • rowData column storing adjusted (corrected for multiple testing) p-values of differential expression between sample groups (named "ADJ.PVAL")

Additional optional annotations:
  • colData column defining paired samples or sample blocks (named "BLOCK")
  • metadata slot named "annotation" giving the organism under investigation in KEGG three letter code (e.g. "hsa" for Homo sapiens)
  • metadata slot named "dataType" indicating the expression data type ("ma" for microarray, "rseq" for RNA-seq)

gs  Gene sets. Either a list of gene sets (character vectors of gene IDs) or a text file in GMT format storing all gene sets under investigation.

grn  Gene regulatory network. Either an absolute file path to a tabular file or a character matrix with exactly *THREE* cols; 1st col = IDs of regulating genes; 2nd col = corresponding regulated genes; 3rd col = regulation effect; Use '+' and '-' for activation/inhibition.

prune.grn  Logical. Should the GRN be pruned? This removes duplicated, self, and reversed edges. Defaults to TRUE.

alpha  Statistical significance level. Defaults to 0.05.

perm  Number of permutations of the expression matrix to estimate the null distribution. Defaults to 1000. If using method='ggea', it is possible to set 'perm=0' to use a fast approximation of gene set significance to avoid permutation testing. See Details.

padj.method  Method for adjusting nominal gene set p-values to multiple testing. For available methods see the man page of the stats function `p.adjust`. Defaults to 'none', i.e. leaves the nominal gene set p-values unadjusted.

out.file  Optional output file the gene set ranking will be written to.

browse  Logical. Should results be displayed in the browser for interactive exploration? Defaults to FALSE.

assay  Character. The name of the assay for enrichment analysis if se is a `SummarizedExperiment` with *multiple assays*. Defaults to "auto", which automatically determines the appropriate assay based on data type provided and enrichment method selected. See details.

...  Additional arguments passed to individual nbea methods. This includes currently:
• beta: Log2 fold change significance level. Defaults to 1 (2-fold).

For SPIA and NEAT:

• sig.stat: decides which statistic is used for determining significant DE genes. Options are:
  – 'p' (Default): genes with adjusted p-value below alpha.
  – 'fc': genes with abs(log2(fold change)) above beta
  – '&': p & fc (logical AND)
  – '|' : p | fc (logical OR)
  – 'xxp': top xx % of genes sorted by adjusted p-value
  – 'xxfc' top xx % of genes sorted by absolute log2 fold change.

For GGEA:

• cons.thresh: edge consistency threshold between -1 and 1. Defaults to 0.2, i.e. only edges of the GRN with consistency >= 0.2 are included in the analysis. Evaluation on real datasets has shown that this works best to distinguish relevant gene sets. Use consistency of -1 to include all edges.
• gs.edges: decides which edges of the grn are considered for a gene set under investigation. Should be one out of c('&', '|'), denoting logical AND and OR, respectively. Accordingly, this either includes edges for which regulator AND / OR target gene are members of the investigated gene set.

Details

'ggea': gene graph enrichment analysis, scores gene sets according to consistency within the given gene regulatory network, i.e. checks activating regulations for positive correlation and repressing regulations for negative correlation of regulator and target gene expression (Geistlinger et al., 2011). When using 'ggea' it is possible to estimate the statistical significance of the consistency score of each gene set in two different ways: (1) based on sample permutation as described in the original publication (Geistlinger et al., 2011) or (2) using an approximation in the spirit of Bioconductor’s npGSEA package that is much faster.

'spia': signaling pathway impact analysis, combines ORA with the probability that expression changes are propagated across the pathway topology; implemented in Bioconductor’s SPIA package (Tarca et al., 2009).

'pathnet': pathway analysis using network information, applies ORA on combined evidence for the observed signal for gene nodes and the signal implied by connected neighbors in the network; implemented in Bioconductor’s PathNet package.

'degraph': differential expression testing for gene graphs, multivariate testing of differences in mean incorporating underlying graph structure; implemented in Bioconductor’s DEGraph package.

'topologygsa': topology-based gene set analysis, uses Gaussian graphical models to incorporate the dependence structure among genes as implied by pathway topology; implemented in CRAN’s topologyGSA package.

'ganpa': gene association network-based pathway analysis, incorporates network-derived gene weights in the enrichment analysis; implemented in CRAN’s GANPA package.

'cepa': centrality-based pathway enrichment, incorporates network centralities as node weights mapped from differentially expressed genes in pathways; implemented in CRAN’s CePa package.
'netgsa': network-based gene set analysis, incorporates external information about interactions among genes as well as novel interactions learned from data; implemented in CRAN's NetGSA package.

'neat': network enrichment analysis test, compares the number of links between differentially expressed genes and a gene set of interest to the number of links expected under a hypergeometric null model; proposed by Signorelli et al. (2016) and implemented in CRAN’s neat package.

It is also possible to use additional network-based enrichment methods. This requires to implement a function that takes 'se', 'gs', and 'grn' and as arguments and returns a numeric matrix 'res.tbl' with a mandatory column named 'PVAL' storing the resulting p-value for each gene set in 'gs'. The rows of this matrix must be named accordingly (i.e. rownames(res.tbl) == names(gs)). See examples.

Using a `SummarizedExperiment` with *multiple assays*:

For the typical use case within the EnrichmentBrowser workflow this will be a `SummarizedExperiment` with two assays: (i) an assay storing the *raw* expression values, and (ii) an assay storing the *normalized* expression values as obtained with the `normalize` function.

In this case, `assay = "auto"` will automatically determine the assay based on the data type provided and the enrichment method selected. For usage outside of the typical workflow, the `assay` argument can be used to provide the name of the assay for the enrichment analysis.

Value

nbeaMethods: a character vector of currently supported methods;

nbea: if(is.null(out.file)): an enrichment analysis result object that can be detailedly explored by calling `eaBrowse` and from which a flat gene set ranking can be extracted by calling `gsRanking`. If 'out.file' is given, the ranking is written to the specified file.

Author(s)

Ludwig Geistlinger

References


See Also

Input: `readSE`, `probe2gene`, `getGenesets` to retrieve gene set definitions from databases such as GO and KEGG. `compileGRN` to construct a GRN from pathway databases.

Output: `gsRanking` to rank the list of gene sets. `eaBrowse` for exploration of resulting gene sets.

Other: `sbea` to perform set-based enrichment analysis. `cwbResults` to combine results from different methods.
Examples

# currently supported methods
nbeaMethods()

# (1) expression data:
# simulated expression values of 100 genes
# in two sample groups of 6 samples each
se <- makeExampleData(what="SE")
se <- deAna(se)

# (2) gene sets:
# draw 10 gene sets with 15-25 genes
gs <- makeExampleData(what="gs", gnames=names(se))

# (3) make 2 artificially enriched sets:
sig.genes <- names(se)[rowData(se)$ADJ.PVAL < 0.1]
gs[[1]] <- sample(sig.genes, length(gs[[1]]))
gs[[2]] <- sample(sig.genes, length(gs[[2]]))

# (4) gene regulatory network
grn <- makeExampleData(what="grn", nodes=names(se))

# (5) performing the enrichment analysis
ea.res <- nbea(method="ggea", se=se, gs=gs, grn=grn)

# (6) result visualization and exploration
gsRanking(ea.res, signif.only=FALSE)

# using your own function as enrichment method
dummyNBEA <- function(se, gs, grn)
{
  sig.ps <- sample(seq(0,0.05, length=1000),5)
  insig.ps <- sample(seq(0.1,1, length=1000), length(gs)-5)
  ps <- sample(c(sig.ps, insig.ps), length(gs))
  score <- sample(1:100, length(gs), replace=TRUE)
  res.tbl <- cbind(score, ps)
  colnames(res.tbl) <- c("SCORE","PVAL")
  rownames(res.tbl) <- names(gs)
  return(res.tbl[order(ps),])
}

ea.res2 <- nbea(method=dummyNBEA, se=se, gs=gs, grn=grn)
gsRanking(ea.res2)

normalize

Normalization of microarray and RNA-seq expression data
normalize

Description

This function wraps commonly used functionality from limma for microarray normalization and from EDASEq for RNA-seq normalization.

Usage

normalize(
  se,
  norm.method = "quantile",
  data.type = c(NA, "ma", "rseq"),
  filter.by.expr = TRUE
)

Arguments

se An object of class SummarizedExperiment.

norm.method Determines how the expression data should be normalized. For available microarray normalization methods see the man page of the limma function normalizeBetweenArrays. For available RNA-seq normalization methods see the man page of the EDASEq function betweenLaneNormalization. For microarray data, defaults to 'quantile', i.e. normalization is carried out so that quantiles between arrays/samples are equal. For RNA-seq data, defaults to 'upper', i.e. normalization is carried out so that quantiles between lanes/samples are equal up to the upper quartile. For RNA-seq data, this can also be 'vst', 'voom', or 'deseq2' to invoke a variance-stabilizing transformation that allows statistical modeling as for microarray data. See details.

data.type Expression data type. Use 'ma' for microarray and 'rseq' for RNA-seq data. If NA, the data type is automatically guessed: if the expression values in se are decimal (float) numbers, they are assumed to be microarray intensities; whole (integer) numbers are assumed to be RNA-seq read counts. Defaults to NA.

filter.by.expr Logical. For RNA-seq data: include only genes with sufficiently large counts in the DE analysis? If TRUE, excludes genes not satisfying a minimum number of read counts across samples using the filterByExpr function from the edgeR package. Defaults to TRUE.

Details

Normalization of high-throughput expression data is essential to make results within and between experiments comparable. Microarray (intensity measurements) and RNA-seq (read counts) data exhibit typically distinct features that need to be normalized for. For specific needs that deviate from standard normalizations, the user should always refer to more specific functions/packages. See also the limma's user guide http://www.bioconductor.org/packages/limma for definition and normalization of the different expression data types.

Microarray data is expected to be single-channel. For two-color arrays, it is expected that normalization within arrays has been already carried out, e.g. using normalizeWithinArrays from limma.
RNA-seq data is expected to be raw read counts. Please note that normalization for downstream DE analysis, e.g. with edgeR and DESeq2, is not ultimately necessary (and in some cases even discouraged) as many of these tools implement specific normalization approaches. See the vignette of EDASeq, edgeR, and DESeq2 for details.

Using `norm.method = "vst"` invokes a variance-stabilizing transformation (VST) for RNA-seq read count data. This accounts for differences in sequencing depth between samples and over-dispersion of read count data. The VST uses the `cpm` function implemented in the edgeR package to compute moderated log2 read counts. Using edgeR’s estimate of the common dispersion phi, the `prior.count` parameter of the `cpm` function is chosen as 0.5/phi as previously suggested (Harrison, 2015).

**Value**

An object of class `SummarizedExperiment`.

**Author(s)**

Ludwig Geistlinger

**References**

Harrison (2015) Anscombe’s 1948 variance stabilizing transformation for the negative binomial distribution is well suited to RNA-seq expression data. doi:10.7490/f1000research.1110757.1


**See Also**

`readSE` for reading expression data from file;

`normalizeWithinArrays` and `normalizeBetweenArrays` for normalization of microarray data;

`withinLaneNormalization` and `betweenLaneNormalization` from the EDASeq package for normalization of RNA-seq data;

`cpm`, `estimateDisp`, `voom`, and `varianceStabilizingTransformation` from the DESeq2 package.

**Examples**

```r
# (1) simulating expression data: 100 genes, 12 samples
#

# (a) microarray data: intensity measurements
maSE <- makeExampleData(what="SE", type="ma")

# (b) RNA-seq data: read counts
rseqSE <- makeExampleData(what="SE", type="rseq")
```
# (2) Normalization
#
# (a) microarray ...
maSE <- normalize(maSE)
assay(maSE, "raw")[1:5,1:5]
assay(maSE, "norm")[1:5,1:5]

# (b) RNA-seq ...
normSE <- normalize(rseqSE, norm.method = "vst")
assay(maSE, "raw")[1:5,1:5]
assay(maSE, "norm")[1:5,1:5]

---

**plots**

**Visualization of gene expression**

**Description**
Visualization of differential gene expression via heatmap, p-value histogram and volcano plot (fold change vs. p-value).

**Usage**

```r
pdistr(p)
volcano(fc, p)
exprsHeatmap(expr, grp, scale.rows = TRUE)
```

**Arguments**

- `p` Numeric vector of p-values for each gene.
- `fc` Numeric vector of fold changes (typically on log2 scale).
- `expr` Expression matrix. Rows correspond to genes, columns to samples.
- `grp` *BINARY* group assignment for the samples. Use '0' and '1' for unaffected (controls) and affected (cases) samples, respectively.
- `scale.rows` Should rows of the expression matrix be scaled for better visibility of expression differences between sample groups? Defaults to TRUE.

**Value**
None, plots to a graphics device.
**probe2gene**

Transformation of probe level expression to gene level expression

### Description

Transforms expression data on probe level to gene level expression by summarizing all probes that are annotated to a particular gene.

### Usage

```r
probe2gene(
  probeSE,
  chip = NA,
  from = "PROBEID",
  to = "ENTREZID",
  multi.to = "first",
  multi.from = "mean"
)
```

### Arguments

- **probeSE**: Probe expression data. An object of class `SummarizedExperiment`. Make sure that the `metadata` contains an element named `annotation` that provides the corresponding ID of a recognized platform such as hgu95av2 (Affymetrix Human Genome U95 chip). This requires that a corresponding `.db` package exists (see [http://www.bioconductor.org/packages/release/BiocViews](http://www.bioconductor.org/packages/release/BiocViews)).
Alternatively, the mapping from probe to gene can also be defined in the `rowData` slot via two columns named (i) `PROBEID` for the platform-specific probe ID, and (ii) `ENTREZID` for the corresponding NCBI Entrez Gene ID.

`chip` Character. The ID of a recognized microarray platform. Only required if not provided in the `metadata` of `probeSE` via an element named annotation.

`from` Character. ID type from which should be mapped. Corresponds to the ID type of the names of argument `se`, with the default `PROBEID` being appropriate if the mapping is based on Bioconductor annotation packages. Note that `from` is ignored if `to` is a `rowData` column of `probeSE`.

`to` Character. Gene ID type to which should be mapped. Corresponds to the gene ID type the rownames of argument `probeSE` should be updated with. Note that this can also be the name of a column in the `rowData` slot of `probeSE` to specify user-defined mappings in which conflicts have been manually resolved. Defaults to `ENTREZID`.

`multi.to` How to resolve 1:many mappings, i.e. multiple gene IDs for a single probe ID? This is passed on to the `multiVals` argument of `mapIds` and can thus take several pre-defined values, but also the form of a user-defined function. However, note that this requires that a single gene ID is returned for each probe ID. Default is "first", which accordingly returns the first gene ID mapped onto the respective probe ID.

`multi.from` How to resolve many:1 mappings, i.e. multiple probe IDs mapping to the same gene ID? Pre-defined options include:

- 'mean' (Default): updates the respective gene expression with the average over the expression of all probes mapping to that gene,
- 'first': returns the first probe ID for each gene ID with multiple probe IDs,
- 'minp' selects the probe ID with minimum p-value (according to the `rowData` column `PVAL` of `probeSE`),
- 'maxfc' selects the probe ID with maximum absolute log2 fold change (according to the `rowData` column `FC` of `probeSE`).

Value

A `SummarizedExperiment` on gene level.

Author(s)

Ludwig Geistlinger

See Also

`readSE` for reading expression data from file, `deAna` for differential expression analysis.

Examples

```r
# (1) reading the expression data from file
exprs.file <- system.file("extdata/exprs.tab", package="EnrichmentBrowser")
```
cdat.file <- system.file("extdata/colData.tab", package="EnrichmentBrowser")
rdat.file <- system.file("extdata/rowData.tab", package="EnrichmentBrowser")
probeSE <- readSE(exprs.file, cdat.file, rdat.file)
geneSE <- probe2gene(probeSE)

---

**Description**

The function reads in plain expression data from file with minimum annotation requirements for the colData and rowData slots.

**Usage**

```r
readSE(
  assay.file,
  cdat.file, 
  rdat.file, 
  data.type = c(NA, "ma", "rseq"),
  NA.method = c("mean", "rm", "keep")
)
```

**Arguments**

- **assay.file**
  - Expression matrix. A tab separated text file containing expression values. Columns = samples/subjects; rows = features/probes/genes; NO headers, row or column names.

- **cdat.file**
  - Column (phenotype) data. A tab separated text file containing annotation information for the samples in either *two or three* columns. NO headers, row or column names. The number of rows/samples in this file should match the number of columns/samples of the expression matrix. The 1st column is reserved for the sample IDs; The 2nd column is reserved for a *BINARY* group assignment. Use '0' and '1' for unaffected (controls) and affected (cases) sample class, respectively. For paired samples or sample blocks a third column is expected that defines the blocks.

- **rdat.file**
  - Row (feature) data. A tab separated text file containing annotation information for the features. In case of probe level data: exactly *TWO* columns; 1st col = probe/feature IDs; 2nd col = corresponding gene ID for each feature ID in 1st col. In case of gene level data: the gene IDs newline-separated (i.e. just *one* column). It is recommended to use *ENTREZ* gene IDs (to benefit from down-stream visualization and exploration functionality of the EnrichmentBrowser). NO headers, row or column names. The number of rows (features/probes/genes) in this file should match the number of rows/features of the expression matrix. Alternatively, this can also be the ID of a recognized platform such as 'hgu95av2' (Affymetrix Human Genome U95 chip) or 'ecoli2' (Affymetrix E. coli Genome 2.0 Array).
data.type  Expression data type. Use 'ma' for microarray and 'rseq' for RNA-seq data. If NA, data.type is automatically guessed. If the expression values in the expression matrix are decimal numbers, they are assumed to be microarray intensities. Whole numbers are assumed to be RNA-seq read counts. Defaults to NA.

NA.method  Determines how to deal with NA's (missing values). This can be one out of:

- mean: replace NA by the mean over all samples for one feature at a time.
- keep: do nothing. Missing values are kept (which, however, can then cause several issues in the downstream analysis)

Defaults to 'mean'.

Value

An object of class `SummarizedExperiment`.

Author(s)

Ludwig Geistlinger

See Also

`SummarizedExperiment`

Examples

```r
# reading the expression data from file
assay.file <- system.file("extdata/expres.tab", package="EnrichmentBrowser")
cdat.file <- system.file("extdata/colData.tab", package="EnrichmentBrowser")
rdat.file <- system.file("extdata/rowData.tab", package="EnrichmentBrowser")
se <- readSE(assay.file, cdat.file, rdat.file)
```
sbeaMethods()

sbea(
  method = EnrichmentBrowser::sbeaMethods(),
  se,
  gs,
  alpha = 0.05,
  perm = 1000,
  padj.method = "none",
  out.file = NULL,
  browse = FALSE,
  assay = "auto",
  ...
)

Arguments

method  Set-based enrichment analysis method. Currently, the following set-based enrichment analysis methods are supported: 'ora', 'safe', 'gsea', 'padog', 'roast', 'camera', 'gsa', 'gsva', 'globaltest', 'samgs', 'ebm', and 'mgsa'. For basic ora also set 'perm=0'. Default is 'ora'. This can also be a user-defined function implementing a set-based enrichment method. See Details.

se  Expression dataset. An object of class SummarizedExperiment. Mandatory minimal annotations:
   • colData column storing binary group assignment (named "GROUP")
   • rowData column storing (log2) fold changes of differential expression between sample groups (named "FC")
   • rowData column storing adjusted (corrected for multiple testing) p-values of differential expression between sample groups (named "ADJ.PVAL")

Additional optional annotations:
   • colData column defining paired samples or sample blocks (named "BLOCK")
   • metadata slot named "annotation" giving the organism under investigation in KEGG three letter code (e.g. "hsa" for Homo sapiens)
   • metadata slot named "dataType" indicating the expression data type ("ma" for microarray, "rseq" for RNA-seq)

gs  Gene sets. Either a list of gene sets (character vectors of gene IDs) or a text file in GMT format storing all gene sets under investigation.

alpha  Statistical significance level. Defaults to 0.05.

perm  Number of permutations of the sample group assignments. Defaults to 1000. For basic ora set 'perm=0'. Using method="gsea" and 'perm=0' invokes the permutation approximation from the npGSEA package.

padj.method  Method for adjusting nominal gene set p-values to multiple testing. For available methods see the man page of the stats function p.adjust. Defaults to 'none', i.e. leaves the nominal gene set p-values unadjusted.
out.file  Optional output file the gene set ranking will be written to.
browse Logical. Should results be displayed in the browser for interactive exploration? 
        Defaults to FALSE.
assay Character. The name of the assay for enrichment analysis if se is a SummarizedExperiment 
        with *multiple assays*. Defaults to "auto", which automatically determines the 
        appropriate assay based on data type provided and enrichment method selected. 
        See details.

... Additional arguments passed to individual sbea methods. This includes currently for ORA and MGSA:
        • beta: Log2 fold change significance level. Defaults to 1 (2-fold).
        • sig.stat: decides which statistic is used for determining significant DE genes. 
          Options are:
        – 'p' (Default): genes with adjusted p-value below alpha.
        – 'fc': genes with abs(log2(fold change)) above beta
        – '&': p & fc (logical AND)
        – '|': p | fc (logical OR)
        – 'xxp': top xx % of genes sorted by adjusted p-value
        – 'xxfc' top xx % of genes sorted by absolute log2 fold change.

Details

'ora': overrepresentation analysis, simple and frequently used test based on the hypergeometric 
        distribution (see Goeman and Buhlmann, 2007, for a critical review).
'safe': significance analysis of function and expression, generalization of ORA, includes other test 
        statistics, e.g. Wilcoxon’s rank sum, and allows to estimate the significance of gene sets by sample 
        permutation; implemented in the safe package (Barry et al., 2005).
'gsea': gene set enrichment analysis, frequently used and widely accepted, uses a Kolmogorov-
        Smirnov statistic to test whether the ranks of the p-values of genes in a gene set resemble a uniform 
        distribution (Subramanian et al., 2005).
'padog': pathway analysis with down-weighting of overlapping genes, incorporates gene weights 
        to favor genes appearing in few pathways versus genes that appear in many pathways; implemented 
        in the PADOG package.
'roast': rotation gene set test, uses rotation instead of permutation for assessment of gene set signif-
        icance; implemented in the limma and edgeR packages for microarray and RNA-seq data, respectively.
'camera': correlation adjusted mean rank gene set test, accounts for inter-gene correlations as im-
        plemented in the limma and edgeR packages for microarray and RNA-seq data, respectively.
'gsa': gene set analysis, differs from GSEA by using the maxmean statistic, i.e. the mean of the 
        positive or negative part of gene scores in the gene set; implemented in the GSA package.
'gsva': gene set variation analysis, transforms the data from a gene by sample matrix to a gene 
        set by sample matrix, thereby allowing the evaluation of gene set enrichment for each sample; 
        implemented in the GSA package.
'globaltest': global testing of groups of genes, general test of groups of genes for association with 
        a response variable; implemented in the globaltest package.
'samgs': significance analysis of microarrays on gene sets, extends the SAM method for single genes to gene set analysis (Dinu et al., 2007).

'ebm': empirical Brown's method, combines p-values of genes in a gene set using Brown's method to combine p-values from dependent tests; implemented in the EmpiricalBrownsMethod package.

'mgsa': model-based gene set analysis, Bayesian modeling approach taking set overlap into account by working on all sets simultaneously, thereby reducing the number of redundant sets; implemented in the mgsa package.

It is also possible to use additional set-based enrichment methods. This requires to implement a function that takes 'se' and 'gs' as arguments and returns a numeric vector 'ps' storing the resulting p-value for each gene set in 'gs'. This vector must be named accordingly (i.e. names(ps) == names(gs)). See examples.

Using a SummarizedExperiment with *multiple assays*:

For the typical use case within the EnrichmentBrowser workflow this will be a SummarizedExperiment with two assays: (i) an assay storing the *raw* expression values, and (ii) an assay storing the *norm*alized expression values as obtained with the normalize function.

In this case, assay = "auto" will *auto*matically determine the assay based on the data type pro- vided and the enrichment method selected. For usage outside of the typical workflow, the assay argument can be used to provide the name of the assay for the enrichment analysis.

Value

sbeaMethods: a character vector of currently supported methods;
sbea: if(is.null(out.file)): an enrichment analysis result object that can be detailedly explored by calling eaBrowse and from which a flat gene set ranking can be extracted by calling gsRanking. If 'out.file' is given, the ranking is written to the specified file.

Author(s)

Ludwig Geistlinger

References


See Also

Input: readSE, probe2gene getGenesets to retrieve gene sets from databases such as GO and KEGG.

Output: gsRanking to retrieve the ranked list of gene sets. eaBrowse for exploration of resulting gene sets.

Other: nbea to perform network-based enrichment analysis. combResults to combine results from different methods.
Examples

# currently supported methods
sbeaMethods()

# (1) expression data:
# simulated expression values of 100 genes
# in two sample groups of 6 samples each
se <- makeExampleData(what="SE")
se <- deAna(se)

# (2) gene sets:
# draw 10 gene sets with 15-25 genes
gs <- makeExampleData(what="gs", gnames=names(se))

# (3) make 2 artificially enriched sets:
sig.genes <- names(se)[rowData(se)$ADJ.PVAL < 0.1]
gs[[1]] <- sample(sig.genes, length(gs[[1]]))
gs[[2]] <- sample(sig.genes, length(gs[[2]]))

# (4) performing the enrichment analysis
ea.res <- sbea(method="ora", se=se, gs=gs, perm=0)

# (5) result visualization and exploration
gsRanking(ea.res)

# using your own tailored function as enrichment method
dummySBEA <- function(se, gs)
{
  sig.ps <- sample(seq(0, 0.05, length=1000), 5)
  nsig.ps <- sample(seq(0.1, 1, length=1000), length(gs)-5)
  ps <- sample(c(sig.ps, nsig.ps), length(gs))
  names(ps) <- names(gs)
  return(ps)
}

ea.res2 <- sbea(method=dummySBEA, se=se, gs=gs)
gsRanking(ea.res2)
Index

assays, 13
colData, 8, 13
comb.ea.results (combResults), 2
combResults, 2, 12, 16, 33, 44
compile.grn.from.kegg (compileGRN), 4
compileGRN, 4, 15, 33
cfg.ebrowser (cfgEBrowser), 6
cmp, 36
data.frame, 25
DataFrame, 11, 18
de.an (deAna), 8
deAna, 8, 14, 38, 39
DGEList, 24, 25
download.kegg.pathways (downloadPathways), 10
downloadPathways, 6, 10
ea.browse (eaBrowse), 11
eaBrowse, 3, 4, 11, 16, 21, 33, 44
eBayes, 9, 26
ebrowser, 12
EList, 24, 25
estimateDisp, 36
export, 16
exprsHeatmap (plots), 37
filterByExpr, 8, 35
GeneSetCollection, 18, 22, 23
gene.genesets (getGenesets), 17
gene.kegg.genesets (getGenesets), 17
genGenesets, 15, 17, 33, 44
gene/nbeaMethods, 30
gene.graph (geneGraph), 20
geneGraph, 20
geneGraphLegend (geneGraph), 20
glmQLFit, 9, 26
gs-ranking (eaBrowse), 11
gsea (sbeaMethods), 41
gsRanking, 3, 33, 44
gsRanking (eaBrowse), 11
hist, 38
idMap, 22
idTypes, 18
idTypes (idMap), 22
import, 24
install, 28
isAvailable, 28
kegg.species.code, 14, 15
keggGet, 10
keggLink, 19
keggList, 10, 19
KEGGPathway, 5, 6, 10
keytypes, 23
lmFit, 14
makeExampleData, 29
map.ids (idMap), 22
mapIds, 22, 23, 39
metadata, 38, 39
nbea, 3, 4, 11–13, 15, 17, 21, 29, 44
nbea/nbeaMethods, 30
nbeaMethods, 13, 30
normalize, 9, 14, 26, 33, 34, 44
normalizeBetweenArrays, 14, 35, 36
normalizeWithinArrays, 15, 35, 36
ora (sbeaMethods), 41
p.adjust, 8, 9, 26, 31, 42
parse.genesets.from.GMT (getGenesets), 17
parseKGML, 6, 10
pathwayDatabases, 5, 6
INDEX

pathways, 6
pdistr (plots), 37
plots, 37
probe.2.gene.eset (probe2gene), 38
probe2gene, 15, 33, 38, 44
read.eset (readSE), 40
readSE, 9, 15, 26, 33, 36, 39, 40, 44
requireNamespace, 28
rowData, 9, 14, 22, 23, 39
sbea, 3, 4, 11–13, 15, 17, 29, 33
sbea (sbeaMethods), 41
sbeaMethods, 13, 41
showAvailableCollections (getGenesets), 17
showAvailableSpecies (getGenesets), 17
spia (nbeaMethods), 30
SummarizedExperiment, 8, 9, 13, 14, 21–25, 31, 33, 35, 36, 38, 39, 41–44
TopTags, 25
volcano (plots), 37
voom, 9, 26, 36
writeGMT (getGenesets), 17